STUDY OF SIGNALS AND SYSTEMS IN THE FRAMEWORK OF MARKOV'S CONSTRUCTIVE MATHEMATICAL LOGIC

by
ETHIRAJAN GOVINDA RAJAN

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
JANUARY, 1990

STUDY OF SIGNALS AND SYSTEMS IN THE FRAMEWORK OF MARKOV'S CONSTRUCTIVE MATHEMATICAL LOGIC

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

by

ETHIRAJAN GOVINDA RAJAN

to the

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

JANUARY, 1990

CERTIFICATE

Certified that this work 'STUDY OF SIGNALS AND SYSTEMS IN THE FRAMEWORK OF MARKOV'S CONSTRUCTIVE MATHEMATICAL LOGIC' by Mr. Ethirajan Govinda Rajan has been carried out under my supervision and that this has not been submitted elsewhere for a degree.

1.1.'90

(V.P. SINHA)

PROFESSOR

Department of Electrical Engineering

Indian Institute of Technology, Kanpur

INDIA

ACKNOWLEDGEMENT

I am grateful to my thesis supervisor, Prof. Dr. V. P. Sinha, for the overall guidance provided by him throughout the research work, and to Dr. M. U. Siddiqi for extending the image processing laboratory facilities as and when they were required.

To those colleagues and students of the Electrical Engineering department who have given me encouragement and stimulation, I wish to express my sincere thanks.

I am especially grateful to George varghese and Chinnathambi for their assistance in giving a shape to this thesis.

E. G. RAJAN

TABLE OF CONTENTS

PART - II

PART - II

PART - III

LIST OF TABLES

LIST OF FIGURES

SYNOPSIS

ETHIRAJAN GOVINDA RAJAN
Ph.D

Department of Electrical Engineering
Indian Institute of Technology - Kanpur - India

September, 1989

STUDY OF SIGNALS AND SYSTEMS IN THE FRAMEWORK OF MARKOV'S CONSTRUCTIVE MATHEMATICAL LOGIC

This thesis is concerned with the problem of formulating within the framework of Markov's constructive mathematical logic, certain structural concepts of a theory for signals and systems defined over monoids of finite alphabets. Markov's *normal algorithms* play a dominant role in this theory.

The central idea on which the work reported in this thesis is based is that of treating a signal space as a free monoid of an alphabet, and a system as a normal algorithm over the alphabet.

With signals treated as words from a specific alphabet $\mathcal{A}$, our first step is to consider the realization of various signal processing operations in terms of string manipulating normal algorithms. The signal space in this context is a subset of the free monoid, $\mathcal{A}^*$, of $\mathcal{A}$ (i.e., the monoid consisting of all possible finite words from the alphabet $\mathcal{A}$, together with the associative binary operation of concatenation of words).

A normal algorithm $\mathcal{N}$ over a given alphabet $\mathcal{A}$ is in essence a mapping that recognizes a subset X of words in $\mathcal{A}^*$ and maps it onto another subset Y of words in $\mathcal{A}^*$. With the subsets X and Y treated respectively as input and output signal spaces, our next step is to examine on the lines of the theory of formal languages,

the signal processing normal algorithms as rewriting systems described by a grammar and then as automata.

The third step in our work is concerned with the formulation of a theory for the study of signals and systems in the framework of Markov's constructive mathematical logic. The constructive logic introduced by Markov is built on a heirarchy of languages $\{Я_\alpha\}$, whose main constituyents are 'alphabets', 'words' and 'normal algorithms'. Here the term *theory* refers to a set of constructive logical sentences written in a particular language belonging to this heirarchy $\{Я_\alpha\}$. We introduce in this thesis a theory consisting of five constructive logical sentences in addition to those of the axioms of monoids, for which the class of signal processing normal algorithms constitutes a model that we refer as $C_{\mathfrak{N}}$.

For this model $C_{\mathfrak{N}}$, we present two clusters of results, one pertaining to its first order properties and another pertaining to its higher order properties.

The notion of a first order property of a normal algorithmic signal processing system belonging to the model $C_{\mathfrak{N}}$ is defined here as a property that is expressible by a constructive logical sentence in which no normal algorithm is quantified. This notion is analogous to that of a first order property of a classical algebraic system. For a classical algebraic system A, a characterization theorem due to Lyndon states that if P is a first order property then P is preserved in a homomorphic image of A if and only if P is expressible by a positive sentence of the first order predicate calculus. In this thesis we present an analogous characterization theorem of the Lyndon type in the framework of the languages of $\{Я_\alpha\}$. According to this theorem, a first order property $\mathfrak{P}$ of a normal algorithmic signal processing system is preserved in a homomorphic image of the system if and only if $\mathfrak{P}$ is described by a constructive logical sentence that does not contain the negation symbol.

In the classical system theory, certain properties like 'continuity' and

'connectedness' are called 'higher order properties' because they cannot be described by sentences of the first order predicate calculus. Analogously the notion of a 'higher order property' of the model $C_{\mathfrak{R}}$ refers to a property that cannot be described by a constructive logical sentence without quantifying one or more normal algorithms. In the classical sense, higher order system properties are usually studied with the help of various concepts of general topology. In the case of $C_{\mathfrak{R}}$, however, these topological notions are not very appropriate to use because of the fact that they are primarily meant for infinite spaces, whereas the basic spaces of our concern in the study of the model $C_{\mathfrak{R}}$ are finite. We find that it is more appropriate in our case to use Hammer's extended topology. Following Hammer, we introduce the concept of *constructive extended filters* and discuss their relationships with normal algorithms.

The major results of the thesis may be summarized as follows:

(i) With signals represented as words from certain finite alphabets consisting of non-numerical symbols, we demonstrate a technique by which some of the traditional signal processing operations, such as *cyclic shifting* and *linear convolution* could be implemented in terms of string manipulating normal algorithms. Further, with signal processing operations in mind, we propose a particular basis set of elementary substitution formulas using which any normal algorithm can be constructed.

(ii) Fitting has outlined a string manipulation language EFS(str(L)) based on the notion of 'Elementary Formal Systems (EFS)' due to Smullyan. On the same lines, we present another string manipulation language EFS(spl($\mathcal{A}$)) which is suitable for signal processing purposes. In this context, we show that a signal processing operation which is realized by means of a system of normal algorithms, could also be realized by a system of 'procedures' in EFS(spl($\mathcal{A}$)).

Further, with signals treated as languages of a free monoid, we introduce a

grammar called *M-grammar* that finitely specifies potentially infinite subsets (languages) of the free monoid. This M-grammar formalizes a specific type of rewriting systems for normal algorithms just as the phrase structure grammar of the Chomsky heirarchy does for Turing machines. A rewriting system corresponding to a Turing machine consists of a finite set of letter-to-letter substitutions of semi-Thue type that operate on Post words. Similarly, we replace every word-to-word substitution formula of a normal algorithm by a finite set of letter-to-letter semi-Thue substitutions that operate on Post word of specific kind. Since the resulting system of semi-Thue substitutions is essentially a Post-Turing rewriting system obtained from end justified substitution formulas of the normal algorithm, we have chosen to call it *End justified Post-Turing* (EPT) *rewriting system*. Using the notion of EPT rewriting systems, we outline a procedure for the realization of normal algorithms by Turing machines and of Turing machines by normal algorithms.

(iii) The notions of automata and codes are closely associated with each other in the sense that, a subset of a free monoid is not only known as a language, but also as a code, and the finite state machine which recognizes it is known as an automaton. With this in mind, we show that the transcriptions i.e., the coded forms of normal algorithms are strings consisting of words from a *thin biprefix code* set $X = 0 1 1^* 0$, which is a subset of the free monoid $\mathcal{A}_0^*$, where $\mathcal{A}_0 = \{ 0 \ 1 \}$. In addition, we introduce a special class of automata termed as *Cyclic Normal Automata*, that characterize EPT rewriting systems. Our study of automata in this connection has led us to formulate the *principle of normalization of automata* as an alternative version of Markov's 'principle of normalization of algorithms'.

(iv) In the framework of a constructive logic which is built on the system of languages $(\mathcal{A}_\alpha)$, we present a constructive theory Th($\Re$) for the structural study of signals and systems. Th($\Re$) consists of five constructive logical sentences. The sentence is a version of Markov's 'principle of constructive choice'. The

second sentence states that if a signal processing algorithm is applicable to a signal representation then the process of applying the normal algorithm terminates. The third sentence describes the functional equivalence between two signal processing normal algorithms having identical input-output relationships. The fourth sentence is about the admissibility of the operation of composition of normal algorithms, and the fifth is about the admissibility of their union. These five sentences together describe the essential characteristics of a class of normal algorithmic sign·l processing systems belonging to the model $C_{\mathfrak{R}}$.

Within the theory Th($\mathfrak{R}$), we establish a homomorphism theorem analogous to one given by Lyndon for classical systems. Let us consider two signal processing systems $\mathfrak{R}_1$ and $\mathfrak{R}_2$ belonging to the model $C_{\mathfrak{R}}$, and let Δ_1 and Δ_2 denote the two sets of constructive sentences corresponding to first order properties of the two systems respectively. Further, let $\mathcal{L}F$ denote the set of constructive sentences of $\Delta_1 \cap \Delta_2$ that do not contain negation. Then our *homomorphism theorem* asserts that every constructive sentence in $\mathcal{L}F$ that holds for $\mathfrak{R}_1$ also holds in $\mathfrak{R}_2$.

(v) For the study of the model $C_{\mathfrak{R}}$, we formulate the notion of a *constructive extended filter* analogous to that of an *extended filter* given by Hammer. Hammer's work deals with classical sets whereas we are concerned with constructive sets. A point to note here is that the concept of a set is not unique in constructive mathematics, for, it depends on the language chosen from $\{\mathcal{A}_\alpha\}$ to interpret the concept. Following Shanin, we interpret the notion of a constructive set in the following manner: A set is decided by a property, and a set property is described by a special type of formula known as one-parameter formula from a language of $\{\mathcal{A}_\alpha\}$. We outline procedures for constructing two specific types of normal algorithms, where a normal algorithm of the first type decides the membership of an element with reference to a given set property, and a normal algorithm of the second type eliminates duplication of elements in the set. Normal

algorithms of these two types are conjointly represented as a word from a specific alphabet. The transcription of this word in association with the one-parameter formula that describes the set property is viewed here as a constructive set. Based on this notion of a constructive set, we describe a technique of formulating different types of extended filters over constructive sets. We show that a normal algorithm over an alphabet $\mathcal{A}$ is an ordered sequence of pairs of extended filter bases over a set of words from $\mathcal{A}$. Normal algorithms are thus seen to have a direct relevance in all potential applications of Hammer's topological techniques in signal processing. Erlandson has defined the following measures for extended filters: (1) ambiguity, (2) discrimination, (3) resolution and (4) distance between two filters. The same measures are redefined here in terms of normal algorithmic operators mapping a metric lattice of extended filters over a constructive set into the metric space of constructive real numbers.

PART - I

SECTION 1

NONNUMERICAL REPRESENTATION AND PROCESSING OF SIGNALS IN MONOIDS

1.1 INTRODUCTION

The term *Signal Processing*, as it is ordinarily interpreted, refers to all those manipulations on numerical representations of signals that are carried out using numerical algorithms. In this thesis we are concerned with an alternative interpretation in which attention is focussed on symbolic, or nonnumeric, representations of signals, and on their processing using symbol manipulating algorithms.

The motivating factor for the choice of such an interpretation is that there would seem to open up through it, interesting possibilities of relating the area of signal processing to developments in other areas such as those of abstract measurement theory and computable real analysis.

Following ideas drawn from abstract measurement theory [71], and quantum logic [25], [52], it has been suggested [72] that the notion of a signal needs to be looked at in terms of a triple $\langle \tilde{X}, X, \phi \rangle$. In this triple, $\tilde{X}$ is an algebraic or relational structure whose primitives and postulates are decided by empirical considerations about the physical phenomena the signals of interest characterize, and X is a numerical structure, called a representation of $\tilde{X}$, which is a homomorphic image of $\tilde{X}$ through the mapping ϕ. Formulation of X is generally carried out using tools of logic and formal languages. In the case of quantum logic, for instance, it is a logic of propositions characterizing what are called 'yes-no experiments'. The representation X on the other hand is formulated using tools of Analysis, and may consist of just the real line treated as a vector space,

or may even be a lattice of subspaces of an appropriate Hilbert space. If signals are to be studied in this light, then the nonnumerical interpretation adopted here is going to be of considerable significance.

It acquires added significance if we take into account the fact that there has emerged in recent years the area of *Computable Analysis* [1], [8] as an alternative to real Analysis, in which the place of real numbers is taken over by computable real numbers, i.e., those real numbers whose arbitrarily precise rational approximations can be be obtained using computer programs or algorithms. For signal processing operations and algorithms set within the framework of computable analysis, symbolic interpretations would seem to be of central interest.

In general, given a string of symbols one can transform it into another desired string by systematically rearranging or rewriting the symbols contained in the given string according to a finite set of rules. A prescription which gives a set of such rewriting rules along with the details of how they should be applied, is known as a *string manipulating algorithm*. The concept of a *normal algorithm* refers to one such string manipulating algorithm. Essentially, a normal algorithm is an ordered list of a finite number of what are called *substitution formulas*. A substitution formula is analogous to a semi-Thue production of Chomsky type-0 grammar.

1.2 SCOPE OF THE THESIS

In this thesis, three basic issues concerning the study of nonnumerical signal processing systems in terms of string manipulating normal algorithms are considered. They are:

(i) Is it possibile to implement traditional numerical signal processing operations with the help of normal algorithms ?

(ii) A set X of nonnumerical signal representations (i.e., a set X of strings of symbols from an alphabet, say, $\mathcal{A}$) could also be called a *language* because X is a subset of the free monoid $\mathcal{A}^*$ (i.e., the monoid consisting of all possible words from the alphabet $\mathcal{A}$, together with the associative binary operation of concatenation of words). Considering that rewriting systems defined by a grammar, as also automata, are language recognizers, can they be used to describe signal processing normal algorithms ?

(iii) What re the various properties of normal algorithmic signal processing systems ?

This thesis consists of a total number of ten sections, under three parts and each part deals with one of these three basic issues.

The first part consists of sections 1, 2, 3 and 4. In section-2, we review certain relevant details about the notion of a normal algorithm. In section-3, we provide a few string manipulating techniques which would be useful in constructing normal algorithms for implementing any nonnumerical signal processing operation. In section-4, we actually demonstrate the technique of implementing operations such as *cyclic shifting* and *linear convolution of nonnegative integer sequences* by means of certain normal algorithms constructed over specific alphabets.

The second part consists of sections 5 and 6. In this part, we take up the second basic issue of describing normal algorithmic signal processing systems as rewriting systems defined by a grammar and as automata. In the theory of formal languages, a set of words from an alphabet is defined as a language and any subset of it as a sublanguage. According to this definition, a nonnumerical representation of a signal over an alphabet could be treated as a language. Here we interpret the operational rule of a normal algorithm in a slightly different way and show that for any given normal algorithm one can obtain a functionally equivalent Turing machine. By this we are able to formulate a type-0 grammar of

Chomsky type which defines the class of all normal algorithms over an alphabet as serial rewriting systems. In section-6, we introduce the notion of a *Cyclic Normal Automaton* for normal algorithms and demonstrate with an example how a cyclic normal automaton over a nonnumerical alphabet recognizes a language corresponding to a signal space.

The remaining sections 7, 8, 9 and 10 form the third part of this thesis. In this part, we consider a structure $C_{\mathfrak{R}} = \langle \mathcal{N}_{\mathcal{A}} , \circ , \simeq \rangle$ (where $\mathcal{N}_{\mathcal{A}}$ denotes a class of normal algorithms over $\mathcal{A}$, $\circ$ denotes the operation of composition and $\simeq$ denotes the relation of functional equivalence) which finitely specifies nonnumerical signal processing operations and we develop a theory for it in a language of constructive logic. In addition, we provide constructive interpretations to (i) Lyndon-Keisler homomorphism theorem and (ii) Thampuran's extended topological filters and show that the properties of a $C_{\mathfrak{R}}$-type signal processing system could be studied with the help of either of these two. Section-7 provides a theory denoted by Th($\mathfrak{R}$), for the structure $C_{\mathfrak{R}}$ and a short list of properties of $C_{\mathfrak{R}}$-type systems. In section-8, we consider Lyndon-Keisler homomorphism theorem and its improved version by Fujiwara. We present a charaterization theorem which is analogous to Fujiwara's version of the homomorphism theorem for $C_{\mathfrak{R}}$-type systems, in a language of constructive logic. In section-9, we reformulate the notion of an extended filter in constructive terms (the resulting filter is called a *constructive extended filter*) and establish a connection between the notions of a normal algorithm and a constructive extended filter. In section-10, we study, on the lines of Erlandson, certain computational aspects of extended topological filters in terms of quantifiable measures such as (i) ambiguity of a filter, (ii) discrimination of a filter, (iii) resolution of a filter and (iv) distance between two filters. In so doing, we describe these measures in terms of normal algorithmic operators acting from a metric lattice of constructive

extended filters into the metric space of constructive real numbers.

1.3 SIGNALS, SYSTEMS, MONOIDS AND $\mathcal{J}$-CLASS RELATIONS

While the precise definitions of various terms to be used in the thesis will be introduced in the text as the need arises, it is appropriate to clarify here itself as to how we intend to interpret the two basic terms - signals and systems. By a *signal* we mean a word from a free monoid $\mathcal{A}^*$ of an alphabet $\mathcal{A}$ (i.e., the monoid consisting of all possible words from the alphabet $\mathcal{A}$ and the identity element Λ that is also known as the null string, together with the associative binary operation of concatenation of words). By a *signal space* we mean the free monoid $\mathcal{A}^*$ itself or a subset of it. By a *signal processing system* we understand a binary relation on $\mathcal{A}^*$, with the first members of the ordered pairs constituting this relation called *inputs* and the second members called *outputs*.

An alternative way of looking at systems is that in terms of four equivalence relations introduced by Green [26], which hold for semigroups as well as free monoids [26]. These are:

(i) Right or $\mathcal{R}$-class congruences:

$$a, b \in \mathcal{A}^*, \quad a\mathcal{R}b \quad \text{if} \quad a\mathcal{A}^* = b\mathcal{A}^*$$

(ii) Left or $\mathcal{L}$-class congruences:

$$a, b \in \mathcal{A}^*, \quad a\mathcal{L}b \quad \text{if} \quad \mathcal{A}^*a = \mathcal{A}^*b$$

(iii) $\mathcal{J}$-class congruences:

$$a, b \in \mathcal{A}^*, \quad a\mathcal{J}b \quad \text{if} \quad \mathcal{A}^*a\mathcal{A}^* = \mathcal{A}^*b\mathcal{A}^*$$

(iv) $\mathcal{H}$-class congruences:

$$a, b \in \mathcal{A}^*, \quad a\mathcal{H}b \quad \text{if} \quad a\mathcal{R}b \text{ and } a\mathcal{L}b$$

DEFINITION 1.2.1

An alphabet $\mathcal{A}$, together with a finite set $\mathfrak{F}$ of $\mathfrak{J}$-class equivalence relations is called an *$\mathfrak{J}$-class presentation* of the free monoid $\mathcal{A}^*$. Similarly one can obtain four more types of presentations based on the remaining four equivalence relations $\mathfrak{D}$, $\mathfrak{R}$, $\mathfrak{L}$ and $\mathfrak{H}$.

We may now say that a system is an *$\mathfrak{J}$-class presentation*.

1.4 STUDY OF SYSTEMS IN TER S OF ASSOCIATIVE CALCULI

With systems treated in this manner, we find the notion of an *Associative Calculus* of Markov [23] to be of immediate use because of the fact that this notion has a direct correspondence with the notion of an $\mathfrak{J}$-class presentation of a free monoid. The correspondence between these two notions is described below.

Let us consider an alphabet $\mathcal{A}$, and a symbol $\longleftrightarrow$ which is not in $\mathcal{A}$. Let 'a' and 'b' be two strings of symbols from the alphabet $\mathcal{A}$. Now, the string $a \longleftrightarrow b$ from $\mathcal{A} \cup \{ \longleftrightarrow \}$ is called a *defining formula* in the theory of associative calculus [23]. By virtue of this formula one can transform a given string into another in the following manner: If the given string is 'uav' it is transformed into 'ubv' where u and v are two more strings in the free monoid $\mathcal{A}^*$. On the other hand, if the given string is 'ubv' then it is transformed into 'uav'. A string (sequence of symbols) from an alphabet is also called a *word*. Now, one can clearly see from the above technique of transforming a word into another by virtue of a defining formula, that a defining formula is one kind of interpretation of an $\mathfrak{J}$-class equivalence relation. For example, the defining formula $a \longleftrightarrow b$ over an alphabet $\mathcal{A}$ can be seen to have a direct correspondence with the equivalence relation $a\mathfrak{J}b$ in the free monoid $\mathcal{A}^*$. A finite set of such defining formulas ($\mathfrak{J}$-class relations) over an alphabet $\mathcal{A}$ is called a *defining system*. Let us denote a defining system by

the symbol $\mathfrak{D}$.

ASSOCIATIVE CALCULUS

Now, the couple $\langle \mathcal{A}, \mathfrak{D} \rangle$ is what is known in Markov's terms as an *Associative Calculus*. Let us denote an associative calculus over an alphabet $\mathcal{A}$ by the symbol $\mathfrak{R}$ such that $\mathfrak{R} = \langle \mathcal{A}, \mathfrak{D} \rangle$.

An associative calculus $\mathfrak{R}$ over an alphabet $\mathcal{A}$ leads to operations on certain words in $\mathcal{A}^*$ not by means of prescription, that is, to act precisely in such and such way using the 3-class relations of the defining system $\mathfrak{D}$ in such and such sequence; but by means of permission, that is, to apply the relations without imposing on them any limitations in their choice and their repetitive use.

An associative calculus $\mathfrak{R}$ is said to be applicable to a word P in the free monoid $\mathcal{A}^*$ only when the word P is transformed into another, say Q, after an exhaustive use of the defining system $\mathfrak{D}$. In such a case, the word P is said to be equivalent to the word Q in $\mathfrak{R}$ which is denoted by $\mathfrak{R}$: $P \perp\!\!\perp Q$. Thus an associative calculus $\mathfrak{R}$ over an alphabet $\mathcal{A}$ generates a system of words $S_{\mathfrak{R}}$ which are equivalents with respect to its defining system $\mathfrak{D}$. The system of words $S_{\mathfrak{R}}$ along with the associative binary operation of concatenation $\circ$, forms a mathematical structure $K = \langle S_{\mathfrak{R}} , \circ \rangle$ where K is called an *Associative System*.

> Thus, with the free monoid $\mathcal{A}^*$ being treated as the non-numerical formal structure of a signal space, an associative calculus over the alphabet $\mathcal{A}$ admits of being treated as a nonnumerical signal processing system.

In other words, with a signal space over an alphabet $\mathcal{A}$, one can construct a suitable associative calculus over $\mathcal{A}$ in order to carry out a desired non-numerical signal processing operation by symbol manipulation.

1.4.1 UNSOLVABLE WORD PROBLEMS IN THE USE OF ASSOCIATIVE CALCULI

In spite of its neat formalism, the theory of associative calculi and systems suffers from various word problems. One of the problems is about the existence of an algorithm which would ascertain the equivalence of a word Q to another word P in an associative calculus. Another problem is about the existence of an algorithm that would precisely recognize an invariant property of an associative system. The first problem has been negatively solved independently by Post and Markov [21]. Ma kov has demonstrated the unsolvability of the second problem [23].

All these amount to saying that one cannot conveniently carry out various non-numerical signal processing operations in terms of associative calculi, and it is mainly because of the following two reasons:

(i) All the word problems of the theory of associative calculus were found to be tied up with the need for a precise definition of the notion of an *algorithm* which is closely linked with the notion of *computability*.

(ii) Monoid property of an associative calculus is not a *hereditary property*. The notion of a hereditary property is to be understood in the following manner: Let $\mathfrak{R}_1$ and $\mathfrak{R}_2$ be two (associative) calculi over the alphabets $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. By a homomorphism of $\mathfrak{R}_1$ into $\mathfrak{R}_2$ we mean an algorithm, say $\mathcal{N}$, (precisely known as a normal algorithm; Ref. Section 2 for details) over the union of $\mathcal{A}_1$ and $\mathcal{A}_2$ which transforms each and every word from $\mathcal{A}_1$ into some word from $\mathcal{A}_2$ and for any P, Q $\in S_{\mathfrak{R}_1}$ the condition $\mathcal{N}(PQ) = \mathcal{N}(P)\mathcal{N}(Q)$ holds. The homomorphism becomes an isomorphism if the following condition is satified: $\mathfrak{R}_1$: P $\perp\!\!\perp$ Q $\Leftrightarrow$ $\mathfrak{R}_2$: $\mathcal{N}(P)$ $\perp\!\!\perp$ $\mathcal{N}(Q)$. In other words, two calculi $\mathfrak{R}_1$ and $\mathfrak{R}_2$ over the alphabets $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively, are called *isomorphic* to each other only if the derivability of Q from P in $\mathfrak{R}_1$ implies the derivability of $\mathcal{N}(Q)$ from $\mathcal{N}(P)$ in $\mathfrak{R}_2$ and vice versa. Now let us consider two calculi $\mathfrak{R}_1$ and $\mathfrak{R}_2$. If there exists an isomorphism of $\mathfrak{R}_1$ into $\mathfrak{R}_2$ then $\mathfrak{R}_1$ is said to be *imbeddable* in $\mathfrak{R}_2$. There can be some properties of $\mathfrak{R}_2$ which

are preserved in $\mathfrak{R}_1$. In general, those properties of an associative calculus which are preserved in every one of its imbeddable calculus are known as *hereditary properties*.

Since signal spaces are treated here as monoids, all the invariant properties of systems which are preserved in their homomorphic images cannot be conveniently studied using associative calculi.

So we turn to Markov's notion of a *Normal Algorithm* for symbolic signal processing.

SECTION 2

NORMAL ALGORITHMS

2.1 WHY NORMAL ALGORITHMS ?

In the precise formulation of the concept of an algorithm, notions of recursive functions, Turing machines, Post's working hypothesis and Markov's normal algorithms play equival nt roles.

While the first three of these notions occupy a well-established place in theoretical computer science and other related areas including that of signal processing, the notion of normal algorithms have received very little attention from the point of view of applications in these areas. Potentials for such applications would, however, seem to be enormous, considering the fact that normal algorithms play a key role in the works of Markov [23], Shanin [31], [32] and others on constructive real numbers and Analysis. Our choice of normal algorithms from amongst the equivalent notions just mentioned is based on these considerations.

2.2 WHAT IS MEANT BY A NORMAL ALGORITHM ?

In what follows, we review very briefly that part of the formalism from Markov's monograph which is essential for describing the concept and the working principle of a normal algorithm in general [23].

By an *alphabet* we mean a finite, unordered list of primitive symbols known as letters. For example, $\mathcal{A} = \{ 0 \mid - \}$ denotes an alphabet $\mathcal{A}$ which contains the three letters 0, I and - . Binary operations between alphabets are interpreted in exactly the same way as the set-theoretic operations such as *union, intersection* and *difference* are understood. We shall use the same symbols $\cup$, $\cap$ and \ to denote respectively the union, intersection and the difference operations between

alphabets. Similarly all the relation symbols such as $\subseteq$, $\subset$, $\supseteq$, $\supset$ and other symbols like $\in$ and $\notin$ are also used, which convey the same meanings as they do in set theory.

By a *generic variable*, we mean a variable whose values are the letters taken from an alphabet. We shall use the symbols ξ , η and μ and their subscripted versions to denote generic variables, irrespective of the alphabets over which they range. But, whenever a generic variable is used, we shall define the alphabet over whi h it will range.

A string of letters drawn from an alphabet $\mathcal{A}$ and written one after another is called a *word* from the given alphabet. The word consisting of no letters is called *empty word* or the *null word*, which is denoted by the symbol Λ. The concatenation of two words from an alphabet is also a word from the same alphabet.

DEFINITION 2.2.1 [26]

A word U is called the *left factor* of a word P if the condition UV = P holds for P, where V is another word. In this case, V is called the *right factor* of P. In general, a word V is called a factor of another word P if the condition UVW = P holds for P where U and V are two other words.

In a word of the form UVW, the factors U and W are called the delimiters of the factor V.

Now let us consider an alphabet $\mathcal{A}$ and two other symbols $\longrightarrow$ and $\cdot$ that are not in $\mathcal{A}$. Then words of the types: (i) $P \longrightarrow Q$ and (ii) $P \longrightarrow\cdot Q$ from the alphabet $\mathcal{A} \cup \{\cdot \longrightarrow\}$ are called *substitution formulas*; the former is called a simple substitution formula and the latter a terminal substitution formula. P and Q are words in the free monoid $\mathcal{A}^*$ of the alphabet $\mathcal{A}$, and are known as the left and the right parts of the corresponding formula.

Substitution formulas of the types (i) $\longrightarrow Q$ (ii) $P \longrightarrow$ and (iii) $\longrightarrow\cdot$ whose blank parts are empty words, are admissible formulas, and those of the first

type, we call *injection formulas*.

By application of a substitution formula to a word, we mean the replacement of the left part of the formula that occurs in the given word, by the right part of the formula.

An ordered list of simple and terminal substitution formulas is called a *scheme*. Let us denote such a scheme by the symbol $\mathfrak{G}$. Now, the ordered pair $\langle \mathcal{A}, \mathfrak{G} \rangle$ consisting of a specific alphabet $\mathcal{A}$ and a scheme $\mathfrak{G}$ is known as a *normal algorithm*, denoted in general by the symbol $\mathcal{N}$. By the operation of $\mathcal{N}$ on a word, say P, we mean the application of the formaulas of its scheme to the word P in accordance with the following rules:

(i) If none of the left parts of the substitution formulas of $\mathcal{N}$ is a factor of P, then $\mathcal{N}$ is not applicable to P ; symbolically we say $\neg !\mathcal{N}(P)$.

(ii) If atleast one of the left parts of the substitution formulas of $\mathcal{N}$ is a factor of P, then $\mathcal{N}$ is definitely applicable to P (i.e., P is an input to $\mathcal{N}$) and we say $!\mathcal{N}(P)$.

(iii) If $!\mathcal{N}(P)$, then from the ordered list of $\mathcal{N}$, the first substitution formula that is applicable to P is identified and the right part of the identified formula is substituted for the first occurrence in P this formula's left part. Let the result of this substitution be the word Q. If the applied formula is of the terminal type, the word resulting from the substitution is treated as the desired word transformed by $\mathcal{N}$ and we write $\mathcal{N}$: $P \vdash \cdot Q$. If the applied formula is of the simple type, then the word produced by the substitution is again subjected to $\mathcal{N}$ as outlined in the earlier steps. In this case, we express the one-step transformation as $\mathcal{N}$: $P \vdash Q$. The process of applying $\mathcal{N}$ can stop at some step either naturally (i.e., when no formula could be applied further) or by means of a terminal formula.

In general, instead of the expression $\mathcal{N}$: $P_0 \vdash P_1$, $\mathcal{N}$: $P_1 \vdash P_2$, ... , $\mathcal{N}$: $P_{n-1} \vdash P_n$ we shall use the briefer expressions $\mathcal{N}$: $P_0 \vdash P_1 \vdash P_2 \vdash \ldots \vdash P_n$ or $\mathcal{N}$: $P_0 \models_n P_n$ or

$\mathcal{N}$: $P_0 \models P_n$. Similarly, instead of the expression $\mathcal{N}$: $P_0 \vdash P_1$, $\mathcal{N}$: $P_1 \vdash P_2$, ... , $\mathcal{N}$: $P_{n-1} \vdash \cdot P_n$ we shall use the briefer expressions $\mathcal{N}$: $P_0 \vdash P_1 \vdash P_2 \vdash ... \vdash \cdot P_n$ or $\mathcal{N}$: $P_0 \models_n \cdot P_n$ or $\mathcal{N}$: $P_0 \models \cdot P_n$.

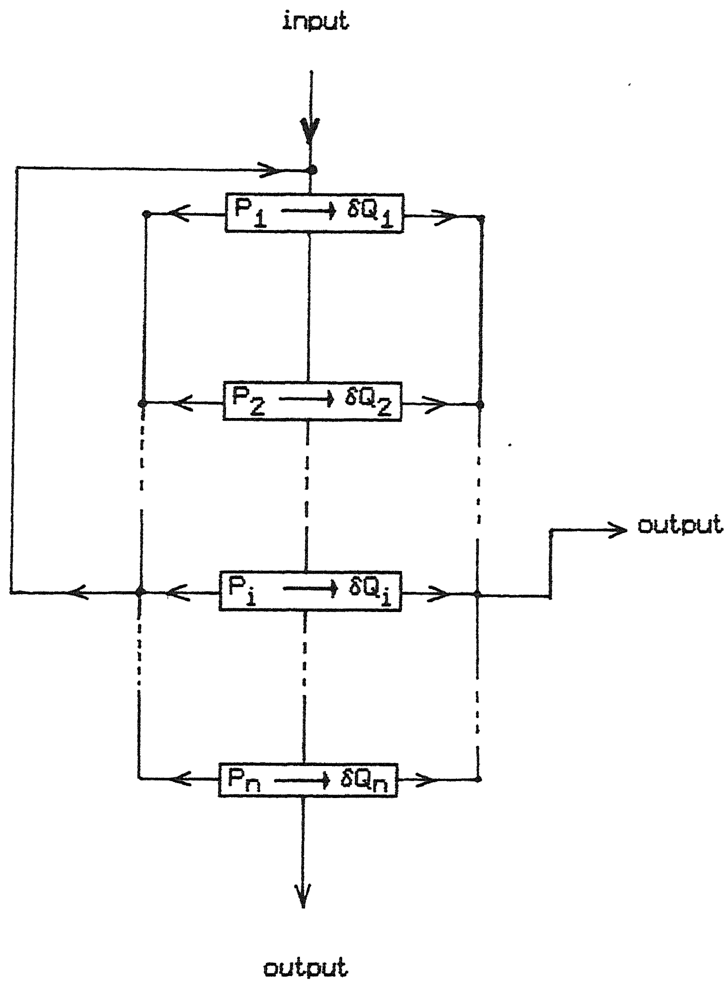


Fig. 2.2.1: Functional block diagram of a normal algorithm

The rules are illustrated by the diagram shown in Fig. 2.2.1; here the symbol δ stands for the empty word Λ if the corresponding formula is simple, and for the symbol $\cdot$ if the formula is a terminal one.

Now, we show that a normal algorithm say $\mathcal{N}$, in an alphabet $\mathcal{A}$ has a direct correspondence with an $\mathfrak{I}$-class presentation of the free monoid $\mathcal{A}^*$, which is defined by certain $\mathfrak{I}$-class partial order relations. The notion of an $\mathfrak{I}$-class presentation of a free monoid has been explained in subsection 1.2. Let a

substitution formula $P \longrightarrow Q$ be contained in the scheme of $\mathcal{N}$. This formula is applicable only to those words of $\mathcal{A}^*$ for which P is a factor. In other words, this substitution formula is a partial map of the type: $\mathcal{A}^*P\mathcal{A}^* \longrightarrow \mathcal{A}^*Q\mathcal{A}^*$ which in turn is interpreted as the partial order relation $P\mathfrak{J}Q$. So, a normal algorithm $\mathcal{N}$ with the scheme $\mathfrak{S}$ consisting of n substitution formulas, is an $\mathfrak{J}$-class presentation of the free monoid $\mathcal{A}^*$ defined by an ordered list of n $\mathfrak{J}$-class partial order relations corresponding to the n substitution formulas.

2.3 HOW TO CONSTRUCT NORMAL ALGORITHMS ?

2.3.1 RELEVANT NOTATIONS AND DEFINITIONS

In theoretical studies dealing with computations, it is customary to assume that there is no scarcity for storage space and there is no restriction on the repeated use of an algorithm. Markov refers to this assumption as the concept of potential realizability [23]. In keeping with this concept, we are allowed to construct alphabets by adding a new letter to every realized alphabet; to construct words by concatenating a letter to every realized word and to construct normal algorithms from ones already constructed. Their realization is *potential*, that is, the process of their construction is not limited by insufficient space and means.

Let $\mathcal{A}$ be an alphabet. Then, the construction of the free monoid $\mathcal{A}^*$ is potentially realizable.

DEFINITION 2.3.1 [26]

Let P be a word in $\mathcal{A}^*$ where $\mathcal{A}$ is an alphabet and μ be a letter in $\mathcal{A}$. Then the length of the word P, that is, the number of symbols required to form P is denoted by $|P|$. The number of occurrences of a letter μ in P is denoted by $|P|_\mu$. Note that $\sum_{\mu \in \mathcal{A}} |P|_\mu = |P|$

DEFINITION 2.3.2 [26]

A word P is called *multilinear* if for every letter μ in the alphabet $\mathcal{A}$ the condition $|P|_{\mu} < 2$ holds for P.

In the study of normal algorithms, there is a need to distinguish between the equivalence of normal algorithms, and that of the words representing them. As we shall see later, different words may represent the same normal algorithm. In order to take care of this distinction, three different types of equivalences are used.

DEFINITION 2.3.3 [23]

Two words P and Q from an alpahbet $\mathcal{A}$ are *graphically equivalent* only when they are composed of the same letters, arranged in the same order. Their equivalence is denoted by $P \doteq Q$, where the symbol $\doteq$ represents the relation of graphical equivalence.

DEFINITION 2.3.4 [20]

Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two normal algorithms in an alphabet $\mathcal{A}$. If $\mathcal{N}_1$ transforms a particular word P from $\mathcal{A}$ into a word Q from $\mathcal{A}$, and $\mathcal{N}_2$ also transforms the same word P into the same word Q, then $\mathcal{N}_1$ and $\mathcal{N}_2$ are said to be *conditionally equivalent* with respect to the word P. Now, if $\mathcal{N}_1$ and $\mathcal{N}_2$ have the same set of input-output pairs consisting of words from the free monoid $\mathcal{A}^*$, then the two normal algorithms are said to be *functionally equivalent* with respect to every admissible input P from $\mathcal{A}^*$ and we express it as $\mathcal{N}_1(P) \simeq \mathcal{N}_2(P)$. At times we use a briefer expression $\mathcal{N}_1 \simeq \mathcal{N}_2$ instead of $\mathcal{N}_1(P) \simeq \mathcal{N}_2(P)$.

DEFINITION 2.3.5 [23]

Let P be a word from an alphabet $\mathcal{A}$. The first symbol in the word P is known as its *initial* and the last symbol in P is known as its *ending*. Any factor of P starting with the initial of P is known as the *head* of P. Similarly, any factor of P ending with the ending of P is known as the *tail* of P. The null string Λ, is an empty word from the alphabet $\mathcal{A}$ and so it cannot be the head or tail of P.

DEFINITION 2.3.6 [23]

Let P be a word from an alphabet $\mathcal{A}$. Then, the word that is obtained by writing the sequence of letters of P in the reverse order is called the *inverse* of the word P and is denoted by P^{-1}.

DEFINITION 2.3.7 [23]

Two words P and Q from an alphabet $\mathcal{A}$ are said to be *relatively prime* to each other if P and Q do not have any common head or tail.

DEFINITION 2.3.8 [23]

Let us consider an alphabet of two symbols, say, $\mathcal{A} = \{ \alpha \ \beta \}$. Then, words of the form $\alpha\beta\alpha$, $\alpha\beta\beta\alpha$, $\alpha\beta\beta\beta\alpha$, and so on, are called *(α,β)-links*. Any word from this alphabet which can be factored into (α,β)-links is called an *(α,β)-chain*.

In general, a normal algorithm $\mathcal{N}$ is called a normal algorithm over an alphabet, say $\mathcal{A}$, if it is constructed in some extension of $\mathcal{A}$ (i.e., in the disjoint union of the alphabets $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{B}$ is called the alphabet of *auxiliary symbols*).

Auxiliary symbols are used as markers that separate desired factors in a word from $\mathcal{A}$.

A normal algorithm meant for a specific operation, say $\bullet$ on certain words from the alphabet $\mathcal{A}$, is denoted by $\mathcal{N}^{\bullet}$. The symbol $\bullet$ may be replaced by any conventional operation symbol or by a suitable acronym which denotes the required operation. At times, it may be required to indicate the domain D of the operation corresponding to $\mathcal{N}^{\bullet}$. Then the normal algorithm is denoted by $\mathcal{N}^{\bullet}_{D}$. The symbol D may be replaced by any symbol denoting a finite or a potentially infinite set of words from the given alphabet. If the domain of an operation of a normal algorithm $\mathcal{N}^{\bullet}$ over an alphabet $\mathcal{A}$ is the free monoid $\mathcal{A}^{*}$ itself, then we shall not, in general, indicate the domain. But if it becomes necessary that we should indicate the domain, then instead of denoting the normal algorithm as $\mathcal{N}^{\bullet}_{\mathcal{A}^{*}}$ we shall denote it as $\mathcal{N}^{\bullet}_{\mathcal{A}}$ in order to reduce the notational complexity. For example, $\mathcal{N}^{\bullet}_{\mathcal{A}_1 \cap \mathcal{A}_2}$ denotes

a normal algorithm that carries out the operation $\bullet$ on words in the free monoid $(\mathcal{A}_1 \cap \mathcal{A}_2)^*$ where $\mathcal{A}_1$ and $\mathcal{A}_2$ are alphabets; wheras, $\mathcal{N}^{\bullet}_{X \cap Y}$ denotes a normal algorithm that carries out the operation $\bullet$ on words that are common to both the sets X and Y of words from the given alphabet.

2.3.2 GUIDE LINES FOR CONSTRUCTING SIMPLE NORMAL ALGORITHMS

In this subsection, we provide certain guidelines that are helpful in the constr‹ction of a normal algorithm meant for a desired operation on words. ..efore that, we present two simple examples of normal algorithms and see how they work.

EXAMPLE: 2.3.2.1

Let us consider the alphabet $\mathcal{A} = \{ a\ b \}$, and the normal algorithm $\mathcal{N}^{ANN}$ whose scheme is as follows:

$\mathcal{N}^{ANN}$:

Substitution formulas	Formula number
$a \longrightarrow$	(0)
$b \longrightarrow$	(1)

The substitution formulas of this scheme convey the meaning that both the letters a and b are to be substituted by null strings Λ if they are found in any given word from $\mathcal{A}$. Since the alphabet $\mathcal{A}$ consists of only two letters a and b, it can be seen that $\mathcal{N}^{ANN}$ annihilates (erases) every word that is obtained from $\mathcal{A}$. So, the construction of an annihilating normal algorithm over an alphabet consisting of n letters, requires n substitution formulas of the type shown above. However, one can simplify the scheme of $\mathcal{N}^{ANN}$ over any arbitrary alphabet $\mathcal{A}$, by making use of the *generic variable* μ as shown below:

$\mathcal{N}^{ANN}$:

Substitution formula		Formula number
$\mu \longrightarrow$	$(\mu \in \mathcal{A})$	(0)

EXAMPLE: 2.3.2.2

Let us take another example of a normal algorithm $\mathcal{N}^{RAN}$, that right-adjoins (concatenates) a particular word say Q, to any other word, say P, from a given alphabet $\mathcal{A}$. In this example we observe the significant use of an auxiliary symbol.

$\mathcal{N}^{RAN}$:

Substitution formula		Formula number
$\alpha\mu \longrightarrow \mu\alpha$	$(\mu \in \mathcal{A},$ $\alpha \notin \mathcal{A})$	(0)
$\alpha \longrightarrow\cdot\ Q$	$(Q \in \mathcal{A}^{*})$	(1)
$\longrightarrow \alpha$		(2)

The working of $\mathcal{N}^{RAN}$ is demonstrated with the help of the following example : Let us consider the alphabets $\mathcal{A}_0 = \{\ 0\ \ 1\ \}$ and $\mathcal{E} = \{\ \alpha\ \}$ and the words P = 0101 and Q = 010 from $\mathcal{A}_0$. $\mathcal{N}^{RAN}$ right adjoins Q to P in the following manner:

First the auxiliary symbol α is left adjoined to the left of P (formula (2)), so that the word 0101 is transformed into α0101. Then α moves to the right through the word (formula (0)) so that the word α0101 is transformed into 0101α after four steps. Now, α is replaced by the word 010 (formula (1)) so that the word 0101α is transformed into the word 0101010 and the process of applying $\mathcal{N}^{RAN}$ terminates.

$\mathcal{N}^{RAN}$ causes the following elementary transformations of P = 0101 :

Sr.No.	Elementary transformations	Formula used
0	0101 (input string)	-
1	α0101	2
2	0α101	0
3	01α01	0
4	010α1	0
5	0101α	0
6	0101010	1 (process terminates)

In principle, any number of auxiliary symbols may be used in a scheme.

However, as Markov has shown in [23], one needs atmost two auxiliary symbols in constructing any type of normal algorithm. This is achieved by means of an operation that Markov calls *translation*. As we shall use this notion of translation later, in section-6, we feel that a brief description of it, here, would be appropriate. With this idea in mind, we recall some details of it from [23].

Let us consider a two-lettered alphabet, say, $\mathfrak{B} = \{\alpha\ \beta\}$ and an alphabet of auxiliary symbols, say, $\mathcal{E} = \{\Gamma_1\ \Gamma_2 \ldots \Gamma_n\}$ in addition to the basic alphabet $\mathcal{A}$. Let $\mathfrak{C} = \mathcal{A} \cup \mathfrak{B}$ and $\mathfrak{G} = \mathcal{A} \cup \mathcal{E}$ and ς be the generic variable that ranges in $\mathfrak{G}$. Then the translation of a letter ς from the alphabet $\mathfrak{G}$ is defined as (i) $[\varsigma]^T = \varsigma$, if $\varsigma \in \mathcal{A}$; (ii) $[\varsigma]^T = \alpha\beta^i\alpha$, $(1 \leq i \leq n)$ if $\varsigma \in \mathcal{E}$. In other words, all the k letters from the alphabet $\mathcal{E}$ are represented as (α,β)-links from the two-lettered alphabet $\mathfrak{B}$. Now, the translation of a word $P = \varsigma_1\varsigma_2 \ldots \varsigma_i \ldots \varsigma_k$ from the alphabet $\mathfrak{G}$ is obtained by replacing each letter ς_i ; $(1 \leq i \leq k)$ in P, by its corresponding translation. The translation of the word P is then denoted by $[P]^T$. [NOTE: $[\Lambda]^T = \Lambda$]. In the same manner, the translation of a normal algorithm $\mathcal{N}$ over the alphabet $\mathfrak{G}$ is obtained by replacing the left as well as the right parts of all the substitution formulas of the scheme, by their corresponding translations.

Based on these details, we now give a result of Markov, *translation theorem*, which forms one of the central notions of constructive mathematics.

THEOREM 2.3.2.1 [23]

Let a normal algorithm $\mathcal{N}$ be applicable to a word P from an alphabet $\mathcal{A}$. Then, $[\mathcal{N}]^T([P]^T) \simeq [\mathcal{N}(P)]^T$.

Thus, generic variables and auxiliary symbols could be seen to play an important role in the construction of normal algorithms.

Now let us see the guidelines that are useful in constructing a normal algorithm for implementing a desired operation on words.

GUIDELINES

(i) Normal algorithms are applicable only to words from alphabets. So, by computation we mean the transformation of a given word into a desired word [Appendix A.3]. Now, the computation for which a normal algorithm is sought, has to be broken into certain basic symbolic operations similar to what is done to a problem prior to writing a computer program for it.

(ii) The order in which the basic symbolic operations of the computation process are to be carried out, is identified.

(iii) Substitution formulas have to be constructed corresponding to each basic operation of the computation process. For example, let us take the case of constructing a substitution formula corresponding to the basic operation of addition of any two positive integers. We proceed in the following manner. Firstly, we take two positive integers, say 3 and 5, and see how they are added symbolically. The integers 3 and 5 are coded as the words III and IIIII from a one-lettered alphabet $\mathcal{A}$ = {I}. The coded words are then represented as a single string III*IIIII with the auxiliary symbol * acting as a marker. Now if we erase the marker * , the resulting string IIIIIIII corresponds to the integer 8. By this method, the operation of addition of any two positive integers can be carried out symbolically. The substitution formula that carries out this symbolic operation is: $* \longrightarrow\!\!\cdot$.

(iv) Let us assume that the given computation can be carried out by means of a sequence of b basic symbolic operations. As outlined in the previous step, we can construct one or more substitution formulas corresponding to each of the b basic operations. We note that a basic operation on a word transforms it into another word and the transformation due to this basic operation depends on the word that has been transformed by the previous basic operation. Given an ordered sequence of b basic symbolic operations constituting a computation and an input

word P, we exhaust the possibility of carrying out all the other basic operations starting from the b^{th} operation before trying out the first operation and we repeat this procedure for every transformed word. By doing this, we are in a position to maintain the dependency relation in the sequence of basic operations. So, while writing the scheme, the block of substitution formulas corresponding to the b^{th} basic operation has to be written first. Next, the block of substitution formulas corresponding to the $(b-1)^{th}$ basic operation has to be written. This procedure has to be continued till the block of substitution formulas corresponding to the first basic operation is written.

(v) The left parts of the substitution formulas contained in a block might be of different lengths. Now, each block has to be reconstructed with their substitution formulas written in the decreasing order of lengths of their left parts. This is done in order to maintain the dependency relation between the substitution formulas in a block.

The ordered list of all the substitution formulas constructed in the above manner, gives rise to a scheme of the required normal algorithm.

2.3.3 BASIC THEOREMS ON THE COMBINATION OF NORMAL ALGORITHMS

Markov presented in his monograph, a series of theorems that allow the construction of new normal algorithms using certain algorithms that have already been constructed. In view of their importance in constructing normal algorithms, we list these theorems here. For the proofs, the reader is referred to [23].

THEOREM 2.3.3.1

For any pair of normal algorithms $\mathcal{N}_1$ and $\mathcal{N}_2$, constructed over alphabets $\mathcal{A}$ and $\mathcal{B}$ respectively, one can construct a normal algorithm $\mathcal{N}_3$ over the alphabet $\mathcal{C}$, where $\mathcal{C} = \mathcal{A} \cup \mathcal{B}$, such that $\mathcal{N}_3(P) = \mathcal{N}_2(\mathcal{N}_1(P))$ where P is a word from $\mathcal{C}$. This is known as the *composition theorem*.

THEOREM 2.3.3.2

For any pair of normal algorithms $\mathcal{N}_1$ and $\mathcal{N}_2$ constructed over an alphabet $\mathcal{A}$, one can construct a normal algorithm $\mathcal{N}_3$ over $\mathcal{A}$ such that $\mathcal{N}_3(P) = \mathcal{N}_1(P)\mathcal{N}_2(P)$, where P is a word from $\mathcal{A}$. This is known as the *union theorem*.

THEOREM 2.3.3.3

For any normal algorithms $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$ over the alphabets $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ respectively, one may construct a normal algorithm $\mathcal{N}_4$ over the alphabet $\mathcal{G}$, where $\mathcal{G} = \mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$, such that

(i) $\mathcal{N}_4(P) \simeq \mathcal{N}_2(P)$ (P is a word from $\mathcal{G}$ and $\mathcal{N}_3(P) = \Lambda$)

(ii) $\mathcal{N}_4(P) \simeq \mathcal{N}_1(P)$ (P is a word from $\mathcal{G}$ and $\mathcal{N}_3(P) \neq \Lambda$)

This is known as the *branching theorem*.

In many cases, the need arises for the repeated application of a normal algorithm over and over again until the resulting word satisfies a specific condition. In such cases, the following theorem, known as the *repetition theorem* is of use.

THEOREM 2.3.3.4

For any two normal algorithms $\mathcal{N}_1$ and $\mathcal{N}_2$ over the alphabets $\mathcal{A}$ and $\mathcal{B}$ respectively, one may construct a normal algorithm $\mathcal{N}_3$ over the alphabet $\mathcal{C}$ where, $\mathcal{C} = \mathcal{A} \cup \mathcal{B}$ such that $\mathcal{N}_3$ transforms a word P from $\mathcal{C}$ into a word Q if and only if there exists a sequence of words P_0, P_1, P_2,.........., P_n ; ($n \geq 0$) satisfying the following conditions :

(i) $P_0 = P$

(ii) $P_i = \mathcal{N}_1(P_{i-1})$; ($0 < i \leq n$)

(iii) $P_n = Q$

(iv) $\mathcal{N}_2(P_i) \neq \Lambda$; ($0 < i < n$)

(v) $\mathcal{N}_2(P_n) = \Lambda$

To summarise, we have thus arrived at a level of understanding as to (i) why we prefer normal algorithms for our study, (ii) what normal algorithms are, (iii) how normal algorithms over a particular alphabet are interpreted as J-class semi-Thue presentations of the corresponding free monoid and (iv) how we can go about developing a few, in order to meet out any desired computational requirement.

In the next section, we shall see some important techniques using which we can actually construct normal algorithms for implementing desired signal processing operations.

SECTION 3

NORMAL ALGORITHMS FOR IMPLEMENTING NONNUMERICAL SIGNAL PROCESSING OPERATIONS

In this section, we deal with the construction of normal algorithms for certain basic operations of signal processing.

3.1 PRELIMINARIES

The term *string manipulation* is interpreted in a restricted sense, as a rearrangement of letters in a given string consisting of symbols drawn from an alphabet. In general, by string manipulation we mean the transformation of a given string into another desired string by either (i) rearranging the letters contained in the given string in a desired fashion, or (ii) erasing arbitrary symbol(s) in the given string, independent of their position or (iii) introducing arbitrary symbol(s) anywhere in the given string.

In the course of constructing normal algorithms for implementing various traditional signal processing operations, we found that any desired normal algorithmic signal processing system could be constructed with the help of a minimal set of certain basic and advanced string manipulation techniques. The following definitions are required for explaining those techniques :

DEFINITION 3.1.1 [31]

Let us consider an alphabet $\mathcal{A}$, and an extra symbol * which is not in $\mathcal{A}$. Then, a word of the form P*Q from $\mathcal{A}\cup\{*\}$, P,Q $\in \mathcal{A}^*$, is defined as a **-pair of words from* $\mathcal{A}$, P its *left *-term* and Q its *right *-term*. Likewise, with P,Q and R as words from $\mathcal{A}$, P*Q*R is a **-triple of words from* $\mathcal{A}$, Q its *kernel*, P* its *left delimiter* and *R its *right delimiter*. P, Q and R are called **-terms*.

DEFINITION 3.1.2

($\square$*)-*term* is a word of the form $\square P*$ where P is from an alphabet $\mathcal{A}$ and the symbols $\square$ and * are extra symbols which are not in $\mathcal{A}$. We prefer to call the symbols $\square$ and * , delimiters.

DEFINITION 3.1.3

(**)-*term*, or a **-delimited term* is a word of the form $*P*$ where P is from an alphabet $\mathcal{A}$.

DEFINITION 3.1.4 [26]

Let P and Q be two words from an alphabet $\mathcal{A}$, where P can be factored as $P_1P_2P_3......P_n$ and Q as $Q_1Q_2Q_3......Q_n$. Then, the *shuffle* of P and Q is the subset of the free monoid $\mathcal{A}^*$, defined by :

$$P \updownarrow Q \;=\; \{ R \mid R = P_1Q_1P_2Q_2...P_iQ_i...P_nQ_n \;;\; 1 \le i \le n \;;\; n \ge 0 \;;\; P_i \,,\, Q_i \in \mathcal{A}^*;$$

DEFINITION 3.1.5

Let us consider the shuffle $P \updownarrow Q$ where $|P| \neq |Q|$. In such a case, $P \updownarrow Q$ is known as an *improper shuffle*.

DEFINITION 3.1.6

Let P and Q be two words from an alphabet $\mathcal{A}$, where P can be factored as $P_1P_2P_3...P_n$. Then, Q-*dilution of* P is the subset of the free monoid $\mathcal{A}^*$, defined by :

$$P \uparrow Q \;=\; \{ R \mid R = P_1QP_2QP_3.......QP_i........QP_n \;;\; 1 \le i \le n \;;\; n > 1 \;;\; P_i \,,\, Q \in \mathcal{A}^*;$$

With these definitions, we are now ready to examine the following string manipulating techniques:

3.2 BASIC STRING MANIPULATING TECHNIQUES

3.2.1 TRANSPOSITION TECHNIQUE-TYPE 1:

Interchanging the positions of two adjacent symbols in a string is known as *transposition*. As per this technique, we can transpose two <u>specific</u> symbols in a string. For example, let us consider the alphabet of arabic numerals and a

substitution formula 23 $\longrightarrow$ 32. This formula is applicable to the word 105723649 and the transformed word is 105732649.

3.2.2 TRANSPOSITION TECHNIQUE-TYPE 2:

In this technique, we use generic variables along with specific symbols. The purpose of using this technique is to shift a specified symbol in a string towards the left or the right. For example, let us consider the alphabet of arabic numerals and two substitution formulas : F1: $3\mu \longrightarrow \mu 3$ and F2 : $\mu 4 \longrightarrow 4\mu$. μ is a generic variable that ranges over the given alphabet. Let P = 1234567 be a word to which both the formulas F1 and F2 are applicable. If F1 is applied to P just once, the word 1234567 is transformed into the word 1243567. On the other hand if F1 is repeatedly applied, the word 1234567 is ultimately transformed into the word 1245673 and the process of applying F1 is naturally terminated. In the same way, by applying F2 just once, P can be transformed into 1243567 or by applying F2 repeatedly, the word P can be transformed into 4123567.

3.2.3 TRANSPOSITION TECHNIQUE-TYPE 3:

In this technique, we make use of auxiliary symbols along with generic variables. Let us consider an alphabet $\mathcal{A}$ and an auxiliary symbol α which is not in $\mathcal{A}$. Then, substitution formulas of the types: (1) $\alpha\mu \longrightarrow \mu\alpha$ and (2) $\xi\alpha \longrightarrow \alpha\xi$ where μ and ξ are individual generic variables which range in $\mathcal{A}$, constitute this technique. For example, let us refer to subsection 2.3.2 where the scheme of the normal algorithm $\mathcal{N}^{RAN}$ is provided. As per the 2nd formula of the scheme, the auxiliary symbol is introduced in a given string. It is shifted to the right end of the string by means of the repeated application of the 0^{th} formula of the scheme, that is, $\alpha\mu \longrightarrow \mu\alpha$ which falls under this category. One would find a frequent use of this transposition technique while constructing various normal algorithms.

[Note : Formulas of the type $\alpha\beta \longrightarrow \beta\alpha$, where α and β are auxiliary variables, also belong to this category.]

3.2.4 TRANSPOSITION TECHNIQUE-TYPE 4:

Techniques of types 1 to 3 are of use in situations where at least one of the pair of symbols to be transposed is a specific one. There are, however, situations in which we are interested in carrying out certain manipulations such as (1) right adjoining the initial of an arbitrary word to its ending, or (2) left adjoining the ending of a word to its initial. Since the initial and the ending of an arbitrary word are unknown, we cannot specify the corresponding symbols which have to be right (left) shifted.

Therefore, we introduce here transposition technique-type 4, which makes use of substitution formulas of the types (1) $\delta\mu\xi \longrightarrow \xi\delta\mu$, a right shift formula and (2) $\mu\xi\Gamma \longrightarrow \xi\Gamma\mu$, a left shift formula. Here, δ and Γ are auxiliary symbols and μ and ξ are generic variables.

A repeated application of the first formula causes a complete right shift of an arbitrary symbol represented by μ which is to the *right of the shift operator*, that is, the auxiliary symbol δ. Likewise, a repeated application of the second formula causes a complete left shift of an arbitrary symbol represented by ξ which is to the *left of the shift operator*, that is, the auxiliary symbol Γ.

The following examples illustrate the use of this technique :

Let us consider the alphabet $\mathcal{A}$ = { a b c d }, and the alphabet of auxiliary symbols $\mathcal{E}$ = { α δ Γ }. Let μ and ξ be the generic variables that range over $\mathcal{A}$ and P = abcd be a word from $\mathcal{A}$.

EXAMPLE 3.2.4.1: Normal algorithm for right shift operations

$\mathcal{N}^{RS}$: Formula number

$$\delta\mu\xi \longrightarrow \xi\delta\mu \qquad (0)$$

$$\delta \longrightarrow\cdot \qquad (1)$$

$$\longrightarrow \delta \qquad (2)$$

Now, $\mathcal{N}^{RS}$: abcd $\vdash$ δabcd $\vdash$ bδacd $\vdash$ bcδad $\vdash$ bcdδa $\vdash\cdot$ bcda

EXAMPLE 3.2.4.2 : Normal algorithm for left shift operations

$\mathcal{N}^{LS}$:

	Formula number
$\mu\xi\Gamma \longrightarrow \xi\Gamma\mu$	(0)
$\alpha\mu \longrightarrow \mu\alpha$	(1)
$\alpha\alpha \longrightarrow \Gamma$	(2)
$\Gamma \longrightarrow\cdot$	(3)
$\longrightarrow \alpha$	(4)

Now, $\mathcal{N}^{LS}$: abcd $\vdash$ αabcd $\vdash$ aαbcd $\vdash$ abαcd $\vdash$ abcαd $\vdash$ abcdα $\vdash$ αabcdα $\vdash$ aαbcdα $\vdash$ abαcdα $\vdash$ abcαdα $\vdash$ abcd$\alpha\alpha$ $\vdash$ abcdΓ $\vdash$ abdΓc $\vdash$ adΓbc $\vdash$ dΓabc $\vdash\cdot$ dabc

Besides carrying out transformations of symbols in a given string, at times we also need to have some of the symbols removed or erased from them. For this purpose, we introduce here the following annihilating techniques.

3.2.5 ANNIHILATION TECHNIQUE-TYPE 1

By annihilation of a symbol in a word, we mean substitution of that symbol by null string Λ. By annihilating formula-type 1, we mean a substitution formula whose left part is the symbol to be erased and the right part is the null string.

For example, let $\mathcal{A}$ = { a b c d } be an alphabet and μ be a generic variable in $\mathcal{A}$. Then, depending on the need one can incorporate in a scheme, any of the following basic annihilating formulas :

F1 : $\mu \longrightarrow$; F2 : a $\longrightarrow$; F3 : b $\longrightarrow$; F4 : c $\longrightarrow$ and F5 : d $\longrightarrow$.

Let P be any word from $\mathcal{A}$. Then the following statements are valid :

(1) If F1 is applied to P just once, the initial of P will be erased.

(2) If F1 is applied repeatedly, the entire word P will be erased.

(3) If F2 is applied to P just once, the first occurring letter a in P will be erased.

(4) If F2 is applied repeatedly, then $|P|_a = 0$

(5) Statements (3) and (4) are true for other formulas F3, F4, and F5 also.

3.2.6 ANNIHILATION TECHNIQUE-TYPE 2

Annihilating formulas of type 1 cannot, in general, be used for erasing a left factor or a right factor of a given string. To be more precise, let us say that we have to erase the right factor V in a word P = UV. Assume that at least one common symbol occurs both in U and V. Then, in the process of erasing all symbols in V with the help of type 1 annihilating formulas, the common symbol present in U will also get erased.

In order to overcome this difficulty, we provide here annihilating formulas of the following types: F1: $\Box\mu \longrightarrow \Box$ and F2: $\xi\tau \longrightarrow \tau$, where the auxiliary symbols $\Box$ and τ we call *right annihilator* and *left annhilator* respectively. μ and ξ are the generic variables over an alphabet $\mathcal{A}$. Annihilation of the right factor V of a given word P from $\mathcal{A}$, where P = UV, is carried out first by rewriting P as U$\Box$V and then by applying the formula F1 repeatedly till V is erased. Now by applying the type 1 annihilating formula: $\Box \longrightarrow$ to the word U$\Box$ we obtain the word U. Likewise, we can use formula F2 for annihilating the left factr of a word.

The following example illustrates the use of these two annihilators in erasing the initial and the ending of any realizable word from a given alphabet.

EXAMPLE 3.2.6.1.

Let us consider the alphabets $\mathcal{A}$ = { a b } ; $\mathcal{B}$ = { α τ $\Box$ } ; and a word P = abba from $\mathcal{A}$.

$\mathcal{N}^{EXC}$:

		Formula number
$\tau\mu \longrightarrow \mu\tau$	$(\mu \in \mathcal{A})$	(0)
$\alpha\alpha \longrightarrow \Box\tau$		(1)
$\Box\mu \longrightarrow$		(2)
$\xi\tau \longrightarrow\cdot$	$(\xi \in \mathcal{A})$	(3)
$\longrightarrow \alpha$		(4)

Now, $\mathcal{N}^{EXC}$: $abba \vdash \alpha abba \vdash \alpha\alpha abba \vdash \Box\tau abba \vdash \Box a\tau bba \vdash \Box ab\tau ba \vdash$

$\Box abb\tau a \vdash \Box abba\tau \vdash bba\tau \vdash\cdot\, bb$

3.2.7 ANNIHILATION TECHNIQUE-TYPE 3

Finally we introduce a technique for carrying out string manipulations such as: (1) annihilation of alternate symbols in a string, (2) extraction of a word P from its Q-diluted form, that is, from $P \uparrow Q$, (3) truncation of P or Q from the $*$-pair P$*$Q. In such cases, substitution formulas of the following types could be used: F1: $\Box * \mu \longrightarrow \Box *$ and F2: $\xi * \tau \longrightarrow * \tau$

We illustrate the use of these formulas by means of the following examples :

Let us consider the alphabets $\mathcal{A} = \{ a\ b \}$; $\mathcal{B} = \{ * \}$; $\mathcal{C} = \{ \tau\ \theta\ \Box \}$ and the words P = abba and Q = bab from $\mathcal{A}$, such that abba$*$bab is from $\mathcal{A} \cup \mathcal{B}$.

EXAMPLE 3.2.7.1. Normal algorithm for truncating the right $*$-term in a $*$-pair

$\mathcal{N}^{REX}$:

		Formula number
$\Box\mu \longrightarrow \mu\Box$	$(\mu \in \mathcal{A})$	(0)
$\Box * \mu \longrightarrow \Box *$		(1)
$* \longrightarrow$		(2)
$\Box \longrightarrow\cdot$		(3)
$\longrightarrow \Box$		(4)

Now, $\mathcal{N}^{REX}$: abba$*$bab $\vdash$ $\Box$abba$*$bab $\vdash$ a$\Box$bba$*$bab $\vdash$ ab$\Box$ba$*$bab $\vdash$

abb$\Box$a$*$bab $\vdash$ abba$\Box*$bab $\vdash$ abba$\Box*$ab $\vdash$ abba$\Box*$b $\vdash$

abba$\Box*$ $\vdash$ abba$\Box$ $\vdash\cdot$ abba

EXAMPLE 3.2.7.2. Normal algorithm for truncating the left $*$-term in a $*$-pair

$\mathcal{N}^{LEX}$:

		Formula number
$\tau\mu \longrightarrow \mu\tau$	$(\mu \in \mathcal{A})$	(0)
$\xi * \tau \longrightarrow * \tau$	$(\xi \in \mathcal{A})$	(1)
$\tau * \longrightarrow * \tau\theta$		(2)
$* \longrightarrow$		(3)

$$\tau \longrightarrow \qquad (4)$$

$$\theta \longrightarrow \cdot \qquad (5)$$

$$\longrightarrow \tau \qquad (6)$$

Now, $\mathcal{N}^{LEX}$: abba*bab ⊢ τabba*bab ⊢ aτbba*bab ⊢ abτba*bab ⊢

abbτa*bab ⊢ abbaτ*bab ⊢ abba*τθbab ⊢ abb*τθbab ⊢

ab*τθbab ⊢ a*τθbab ⊢ *τθbab ⊢ τθbab ⊢ θbab ⊢· bab

Exclusively with the techniques of transposition, annihilation and injection [subsec on 2.2], we can construct a scheme for any desired normal algoritl n. In what follows, we provide some more techniques which will be of use in constructing normal algorithms for signal processing purposes.

3.3 ADVANCED STRING MANIPULATING TECHNIQUES

3.3.1 CYCLIC PERMUTATIONS IN ANY DESIRED FACTOR OF A WORD :

In general, a rearrangement c a word $P = \xi_1\xi_2...\xi_m$, where $\xi_1,...,\xi_m$ are generic variables in an alphabet, is a word $P' = \xi_{i_1}\xi_{i_2}...\xi_{i_m}$ where $i_1i_2...i_m$ is a permutation of 1234...m. The class of all possible permutations of a word is known as the *rearrangement class* of P, of which *cyclic permutations* of P form a subset [22]. A cyclic permutation in a word can be achieved by means of modified versions of shifting algorithms $\mathcal{N}^{RS}$ or $\mathcal{N}^{LS}$ [Subsection 3.2].

The technique proposed here is concerned with the problem of constructing a normal algorithm that causes cyclic permutations in any desired factor of a given string. We bring out details of this technique with the help of an example.

Let $\mathcal{A}$ be an alphabet and $P = \xi_1\xi_2...\xi_i\xi_{i+1}\xi_{i+2}\xi_{i+3}...\xi_m$ be a word from $\mathcal{A}$, where ξ_i; $1 \le i \le m$ are generic variables in $\mathcal{A}$. Let us say that we are interested in having a cyclic permutation in the factor $\xi_i\xi_{i+1}\xi_{i+2}\xi_{i+3}$ of the word P. To do this, we proceed in two steps: (1) we inject two auxiliary symbols □ and * in P, in such a way that the desired factor becomes a (□*)-term [Definition 3.1.3] and (2) we

introduce certain transposition-type3 formulas in the scheme of the shifting algorithm so that the algorithm is made available only to the (□✱)-term. Note that the auxiliary symbols □ and ✱ act here as delimiters to the desired factor.

For example, the following scheme $\mathcal{N}^{MLS}$, is a modified version of $\mathcal{N}^{LS}$, and it is applicable only to a (□✱)-term :

$\mathcal{N}^{MLS}$: Formula number

$$\mu\xi\Gamma \longrightarrow \xi\Gamma\mu \quad (\mu,\xi \in \mathcal{A}) \qquad (0)$$

$$\alpha\Box \longrightarrow \Box\alpha \qquad (1)$$

$$\alpha\mu \longrightarrow \mu\alpha \qquad (2)$$

$$\alpha\alpha \longrightarrow \Gamma \qquad (3)$$

$$\Gamma \longrightarrow\cdot \qquad (4)$$

$$\longrightarrow \alpha \qquad (5)$$

Let $\mathcal{A} = \{a\ b\}$, $P = abab$ and $P' = a\Box ba{*}b$

Now, $\mathcal{N}^{MLS}$: $a\Box ba{*}b \vdash \alpha a\Box ba{*}b \vdash a\alpha\Box ba{*}b \vdash a\Box\alpha ba{*}b \vdash$

$a\Box b\alpha a{*}b \vdash a\Box ba\alpha{*}b \vdash \alpha a\Box ba\alpha{*}b \vdash a\alpha\Box ba\alpha{*}b \vdash$

$a\Box\alpha ba\alpha{*}b \vdash a\Box b\alpha a\alpha{*}b \vdash a\Box ba\alpha\alpha{*}b \vdash a\Box ba\Gamma{*}b \vdash$

$a\Box a\Gamma b{*}b \vdash\cdot\ a\Box ab{*}b$

Although we have evolved the algorithm $\mathcal{N}^{MLS}$ with reference to a specific factor of a given word, it may be seen that it is of a perfectly general nature. To be more precise, given any word and a factor of it to be cyclically permuted, the formulas of $\mathcal{N}^{MLS}$ can be used.

Sometimes, we come across a situation in which a particular factor of a multifactored string has to be cyclically permuted. For example, let us consider a word P from an alphabet $\mathcal{A}$, consisting of m factors: $P = P_1P_2..P_i..P_m$. We may be interested in getting the i^{th} factor of the string P, to be cyclically permuted. In order to do this, we take the following steps: (i) Firstly, we inject m distinct delimiters $d_1,d_2,d_3,...,d_i,...,d_m$ in such a way that the given string P is expressed

as the shuffle $P \updownarrow d = P_1d_1P_2d_2 \ldots d_{i-1}P_id_i \ldots P_md_m$. (2) Next, we introduce i-1 substitution formulas of transposition-type3, in the scheme of the modified shifting algorithm $\mathcal{N}^{MLS}$ so that the remodified algorithm is made available only to the (d_{i-1},d_i)-term of $P \updownarrow d$.

Now, the following scheme $\mathcal{N}^{CPF}$, causes cyclic permutation in the (d_{i-1},d_i)-term of a shuffle $P \updownarrow d = P_1d_1P_2d_2 \ldots d_{i-1}P_id_i \ldots P_md_m$.

$\mathcal{N}^{CPF}$: Formula number

$\mu\xi\Gamma \longrightarrow \xi\Gamma\mu$		(0)
$\alpha d_1 \longrightarrow d_1\alpha$		(1)
$\vdots$		$\vdots$
$\alpha d_i \longrightarrow d_i\alpha$		(i)
$\alpha\mu \longrightarrow \mu\alpha$		(i+1)
$\alpha\alpha \longrightarrow \Gamma$		(i+2)
$\Gamma \longrightarrow\cdot$		(i+3)
$\longrightarrow \alpha$		(i+4)

Now, $\mathcal{N}^{CPF}(P \updownarrow d) = P_1d_1 \ldots\ldots d_{i-1}P_i'd_i \ldots\ldots P_md_m$, where P_i' is the cyclic permutation of the factor P_i .

We now move on to a further generalization of the shifting algorithm, using which, factors of a given word are separately and simultaneously cyclic permuted.

3.3.2 CYCLIC PERMUTATIONS IN ALL THE FACTORS OF A WORD :

Given a factored string P from an alphabet $\mathcal{A}$, $P = P_1P_2 \ldots P_m$; our problem is to carry out cyclic permutations of all the m factors separately. In order to this, we first of all rewrite the word P as a ✱-diluted string [Definition 3.1.7]. This gives us the string $P\uparrow✱ = P_1✱P_2✱\ldots✱P_m$. We know that cyclic permutation in a string is carried out by a shift operator Γ right adjoined to the string. So, in order to carry out cyclic permutations in all the factors of the given word P, that is, in every ✱-term of the string $P\uparrow✱$, our next step is to inject the shift operator Γ in

$P\dagger*$ as follows: $P_1\Gamma*P_2\Gamma*$.............$\Gamma*P_m\Gamma$. Now the following scheme carries out cyclic permutations in all $*$-terms of $P\dagger*$, one by one from the left, sequentially.

$\mathcal{N}^{FCP}$:

		Formula number
$\xi\alpha\mu\Gamma \longrightarrow \mu\Gamma\xi\alpha$	$(\xi,\mu \in \mathcal{A})$	(00)
$\alpha\mu\Gamma \longrightarrow \mu\Gamma\alpha$		(01)
$\beta\mu \longrightarrow \alpha\mu\beta$		(02)
$\alpha\alpha \longrightarrow \beta$		(03)
$\beta\beta \longrightarrow \Gamma$		(04)
$\Gamma\alpha \longrightarrow \Gamma$		(05)
$\Gamma\mu \longrightarrow \mu\Gamma$		(06)
$\beta\alpha \longrightarrow \beta$		(07)
$\Gamma*\alpha \longrightarrow *\beta$		(08)
$\beta*\mu \longrightarrow \beta*\beta\mu$		(09)
$\Gamma \longrightarrow\cdot$		(10)
$\longrightarrow \alpha$		(11)

Now let us consider an alphabet $\mathcal{A}$ = {a b} , a word P = abba and $P\dagger*$ = ab*ba .

$\mathcal{N}^{FCP}$: $ab*ba \vdash \alpha ab*ba \vdash \alpha\alpha ab*ba \vdash \beta ab*ba \vdash \alpha a\beta b*ba \vdash \alpha a\alpha b\beta*ba \vdash$

$\alpha a\alpha b\beta*\beta ba \vdash \alpha a\alpha b\beta*\alpha b\beta a \vdash \alpha a\alpha b\beta*\alpha b\alpha a\beta \vdash \alpha\alpha a\alpha b\beta*\alpha b\alpha a\beta \vdash$

$\beta a\alpha b\beta*\alpha b\alpha a\beta \vdash \alpha a\beta\alpha b\beta*\alpha b\alpha a\beta \vdash \alpha a\beta b\beta*\alpha b\alpha a\beta \vdash$

$\alpha a\alpha b\beta\beta*\alpha b\alpha a\beta \vdash \alpha a\alpha b\Gamma*\alpha b\alpha a\beta \vdash \alpha b\Gamma a\alpha*\alpha b\alpha a\beta \vdash$

$b\Gamma\alpha a\alpha*\alpha b\alpha a\beta \vdash b\Gamma a\alpha*\alpha b\alpha a\beta \vdash ba\Gamma\alpha*\alpha b\alpha a\beta \vdash$

$ba\Gamma*\alpha b\alpha a\beta \vdash ba*\beta b\alpha a\beta \vdash ba*\alpha b\beta\alpha a\beta \vdash ba*\alpha b\beta a\beta \vdash$

$ba*\alpha b\alpha a\beta\beta \vdash ba*\alpha b\alpha a\Gamma \vdash ba*\alpha a\Gamma b\alpha \vdash ba*a\Gamma\alpha b\alpha \vdash ba*a\Gamma b\alpha \vdash$

$ba*ab\Gamma\alpha \vdash\cdot ba*ab$

The techniques discussed so far (i.e., transposition, annihilation, injection and cyclic permutation), enable us to construct substitution formulas required for different symbol manipulations in words from the union of a basic alphabet $\mathcal{A}$, an

alphabet $\mathcal{S}$ of auxiliary symbols and an alphabet $\mathfrak{D}$ of delimiters. Now, with a formal characterization of substitution formulas, our study of string manipulating techniques would be complete.

3.3.3 PATTERN MATCHING TECHNIQUE :

In general, the operation of examining strings of symbols from an alphabet, for the occurrence of specific strings known as *patterns* is defined as *pattern matching* [17]. As per this definition, a substitution formula, is a prescription for attern matching and substitution. In other words, the left part of a substitution formula, which is a specific pattern, is examined for its occurrence in a given string of symbols and replaced by the right part of the formula, *the value of the pattern*. So, a normal algorithm over an alphabet $\mathcal{A}$, is a prescription for systematically examining a set of strings from the free monoid $\mathcal{A}^*$ for the occurrence of a finite number of patterns and substituting them with their values.

Let us consider a basic alphabet $\mathcal{A}$, an alphabet $\mathfrak{D}$ consisting of delimiters and an alphabet $\mathcal{S}$ consisting of auxiliary symbols. Now, patterns are those strings from $(\mathcal{A} \cup \mathfrak{D} \cup \mathcal{S})^*$ which are used as left parts of various substitution formulas constructed over the union of these disjoint alphabets. We are interested in two kinds of patterns that we call (1) Simple patterns and (2) Complex patterns. A simple pattern is a string either from $\mathcal{A}^*$ or $(\mathcal{A} \cup \mathfrak{D})^*$ that is used as left part of a substitution formula. A complex pattern is a string from $(\mathcal{A} \cup \mathfrak{D} \cup \mathcal{S})^*$ containing at least one symbol from $\mathcal{S}$, and which is used as left part of a substitution formula. Simple and Complex patterns are further divided into *regular simple patterns, irregular simple patterns, regular complex patterns* and *irregular complex patterns*. Regular patterns are those, whose constructions are governed at least by any one of the rules of the string manipulating techniques discussed so far. Irregular patterns are special patterns of both simple and complex types, whose constructions are not governed by any rule, but are used as left parts of

certain substitution formulas.

We now proceed to verify, in the next section, the usefulness of these string manipulating techniques, by actually constructing normal algorithms for carrying out the traditional signal processing operations of cyclic shifting and linear convolution of nonnegative integer sequences.

SECTION 4

REALIZATION OF CERTAIN NORMAL-ALGORITHMIC SIGNAL PROCESSING OPERATIONS

In this section, we demonstrate the use of the string manipulating techniques of section-3, in nonnumerical signal processing. First, we construct a normal algorithm for carrying out *cyclic shifting* in a string of arbitrary length. Next, we outline a general method for constructing normal algorithmic systems for signal processing operations on symbolic sequences in general. As an illustration, we then construct a normal algorithmic system for carrying out linear convolution of nonnegative integer sequences.

4.1 CYCLIC SHIFTING BY MEANS OF A NORMAL ALGORITHM $\mathcal{N}^{CS}$

When the normal algorithm $\mathcal{N}^{LS}$ [Example 3.2.4.2] is applied to a word, say P from an alphabet $\mathcal{A}$, the ending of P is shifted to its left by the shift operator Γ which gets annihilated after the shifting is over. As we shall see later, there are situations in which we make use of Γ not only as a left shift operator, but also as a part of certain patterns to be substituted by their values. In order to do this, there has to be a provision in the scheme of $\mathcal{N}^{LS}$ for bringing Γ back to the right of the word whose ending is left shifted.

With this purpose in mind, the following scheme of $\mathcal{N}^{CS}$ has been constructed over $\mathcal{A}\cup\{\alpha \ \beta \ \Gamma\}$. $\mathcal{N}^{CS}$ causes one cyclic shift (left shift) in any string of arbitrary length, in the free monoid $\mathcal{A}^*$.

$\mathcal{N}^{CS}$: Formula number

$$\xi\alpha\mu\Gamma \longrightarrow \mu\Gamma\xi\alpha \qquad (\xi, \mu \in \mathcal{A}) \qquad (0)$$

$$\alpha\mu\Gamma \longrightarrow \mu\Gamma\alpha \qquad (1)$$

$$\beta\mu \longrightarrow \alpha\mu\beta \qquad (2)$$

$$\alpha\alpha \longrightarrow \beta \qquad (3)$$

$$\beta\beta \longrightarrow \Gamma \qquad (4)$$

$$\Gamma\alpha \longrightarrow \Gamma \qquad (5)$$

$$\Gamma\mu \longrightarrow \mu\Gamma \qquad (6)$$

$$\beta\alpha \longrightarrow \beta \qquad (7)$$

$$\Gamma \longrightarrow\cdot \qquad (8)$$

$$\longrightarrow \alpha \qquad (9)$$

Let us consider, for instance, the alphabets $\mathcal{A} = \{0\ 1\}$ and $\mathcal{B} = \{\alpha\ \beta\ \Gamma\}$ and apply $\mathcal{N}^{CS}$ to a word 1010 from $\mathcal{A}$.

Now, $\mathcal{N}^{CS}$:

Formula number	Elementary transformations
-	1010 ← (Input string)
9	$\alpha 1010$
9	$\alpha\alpha 1010$
3	$\beta 1010$
2	$\alpha 1\beta 010$
2	$\alpha 1\alpha 0\beta 10$
2	$\alpha 1\alpha 0\alpha 1\beta 0$
2	$\alpha 1\alpha 0\alpha 1\alpha 0\beta$
9	$\alpha\alpha 1\alpha 0\alpha 1\alpha 0\beta$
3	$\beta 1\alpha 0\alpha 1\alpha 0\beta$
2	$\alpha 1\beta\alpha 0\alpha 1\alpha 0\beta$
7	$\alpha 1\beta 0\alpha 1\alpha 0\beta$
2	$\alpha 1\alpha 0\beta\alpha 1\alpha 0\beta$
7	$\alpha 1\alpha 0\beta 1\alpha 0\beta$
2	$\alpha 1\alpha 0\alpha 1\beta\alpha 0\beta$
7	$\alpha 1\alpha 0\alpha 1\beta 0\beta$
2	$\alpha 1\alpha 0\alpha 1\alpha 0\beta\beta$
4	$\alpha 1\alpha 0\alpha 1\alpha 0\Gamma$
0	$\alpha 1\alpha 0\alpha 0\Gamma 1\alpha$
0	$\alpha 1\alpha 0\Gamma 0\alpha 1\alpha$
0	$\alpha 0\Gamma 1\alpha 0\alpha 1\alpha$
1	$0\Gamma\alpha 1\alpha 0\alpha 1\alpha$
5	$0\Gamma 1\alpha 0\alpha 1\alpha$
6	$01\Gamma\alpha 0\alpha 1\alpha$
5	$01\Gamma 0\alpha 1\alpha$
6	$010\Gamma\alpha 1\alpha$
5	$010\Gamma 1\alpha$
6	$0101\Gamma\alpha$
5	0101Γ
8	0101 ← (Output string)

$\mathcal{N}^{cs}$ causes one cyclic shift in the word 1010 after 29 elementary transformations such that 1010 becomes 0101. In fact, the number of transformations that a word P of length n undergoes when $\mathcal{N}^{cs}$ is applied to it for one cyclic shift, can be computed with the help of the rule $a+(n-1)d$ where $a=11$ and $d=6$.

It can be observed that the process of applying $\mathcal{N}^{cs}$ to a word P comes to an end after a single cyclic shift, on the application of the 8^{th} substitution formula of the scheme. So, if we require n consecutive cyclic shifts in a word P, $\mathcal{N}^{cs}$ has to be applied consecutively (n-1) number of times to the intermediate strings. But, this can be done in a different manner. Let us replace the terminal formula $\Gamma \longrightarrow \cdot$ of the scheme of $\mathcal{N}^{cs}$ by the simple formula $\Gamma \longrightarrow$ and make use of Theorem 2.3.3.4 in achieving a control over the number of cyclic shifts. Now, the scheme of $\mathcal{N}^{cs}$ with the above modification is labelled as $\mathcal{N}^{ccs}$ where the acronym ccs stands for *Consecutive Cyclic Shifts*. The generality of the scheme of $\mathcal{N}^{ccs}$ has been verified by implementing it in a personal computer, using FORTRAN facility .

For example, if we use the interactive FORTRAN program CYSHFT [Appendix A.1] corresponding to the scheme of $\mathcal{N}^{ccs}$ for carrying out two cyclic shifts in a word $P \doteq 0100111011$ from the alphabet $\mathcal{A} = \{0\ 1\}$, then the output file would contain the following :

```
GIVE THE NO. OF CHARACTERS IN THE INPUT STRING:
  10
GIVE THE INPUT STRING:
 [ Press <return> only after the complete string is given ]
0100111011
SPECIFY NUMBER OF CYCLIC SHIFTS:
2
```

Formula number	Elementary transformations
-	0100111011 (Input string)
9	α0100111011
9	αα0100111011
3	β0100111011
2	α0β100111011
2	α0α1β00111011
2	α0α1α0β0111011
2	α0α1α0α0β111011
2	α0α1α0α0α1β11011
2	α0α1α0α0α1α1β1011
2	α0α1α0α0α1α1α1β011
2	α0α1α0α0α1α1α1α0β11
2	α0α1α0α0α1α1α1α0α1β1
2	α0α1α0α0α1α1α1α0α1α1β
9	αα0α1α0α0α1α1α1α0α1α1β
3	β0α1α0α0α1α1α1α0α1α1β
2	α0βα1α0α0α1α1α1α0α1α1β
7	α0β1α0α0α1α1α1α0α1α1β
2	α0α1βα0α0α1α1α1α0α1α1β
7	α0α1β0α0α1α1α1α0α1α1β
2	α0α1α0βα0α1α1α1α0α1α1β
7	α0α1α0β0α1α1α1α0α1α1β
2	α0α1α0α0βα1α1α1α0α1α1β
7	α0α1α0α0β1α1α1α0α1α1β
2	α0α1α0α0α1βα1α1α0α1α1β
7	α0α1α0α0α1β1α1α0α1α1β
2	α0α1α0α0α1α1βα1α0α1α1β
7	α0α1α0α0α1α1β1α0α1α1β
2	α0α1α0α0α1α1α1βα0α1α1β
7	α0α1α0α0α1α1α1β0α1α1β
2	α0α1α0α0α1α1α1α0βα1α1β
7	α0α1α0α0α1α1α1α0β1α1β
2	α0α1α0α0α1α1α1α0α1βα1β
7	α0α1α0α0α1α1α1α0α1β1β
2	α0α1α0α0α1α1α1α0α1α1ββ
4	α0α1α0α0α1α1α1α0α1α1Γ
0	α0α1α0α0α1α1α1α0α1Γ1α
0	α0α1α0α0α1α1α1α1Γ0α1α
0	α0α1α0α0α1α1α1Γ1α0α1α
0	α0α1α0α0α1α1Γ1α1α0α1α
0	α0α1α0α0α1Γ1α1α1α0α1α
0	α0α1α0α1Γ0α1α1α1α0α1α
0	α0α1α1Γ0α0α1α1α1α0α1α
0	α0α1Γ1α0α0α1α1α1α0α1α
0	α1Γ0α1α0α0α1α1α1α0α1α
1	1Γα0α1α0α0α1α1α1α0α1α
5	1Γ0α1α0α0α1α1α1α0α1α
6	10Γα1α0α0α1α1α1α0α1α
5	10Γ1α0α0α1α1α1α0α1α
6	101Γα0α0α1α1α1α0α1α

5	I0IΓ0α0α Iα Iα Iα0α Iα	
6	I0I0Γα0α Iα Iα Iα0α Iα	
5	I0I0Γ0α Iα Iα Iα0α Iα	
6	I0I00Γα Iα Iα Iα0α Iα	
5	I0I00Γ Iα Iα Iα0α Iα	
6	I0I00IΓα Iα Iα0α Iα	
5	I0I00IΓ Iα Iα0α Iα	
6	I0I00IIΓα Iα0α Iα	
5	I0I00IIΓ Iα0α Iα	
6	I0I00IIIΓα0α Iα	
5	I0I00IIIΓ0α Iα	
6	I0I00III0Γα Iα	
5	I0I00III0Γ Iα	
6	I0I00III0IΓα	
5	I0I00III0IΓ	
8	I0I00III0I	← [First cyclic shift is over after
9	αI0I00III0I	65 elementary transformations]
9	ααI0I00III0I	
3	βI0I00III0I	
2	α Iβ0I00III0I	
2	α Iα0βI00III0I	
2	α Iα0α Iβ00III0I	
2	α Iα0α Iα0β0III0I	
2	α Iα0α Iα0α0βIII0I	
2	α Iα0α Iα0α0α IβII0I	
2	α Iα0α Iα0α0α Iα !βI0I	
2	α Iα0α Iα0α0α Iα Iα Iβ0I	
2	α Iα0α Iα0α0α Iα Iα Iα0βI	
2	α Iα0α Iα0α0α Iα Iα Iα0α Iβ	
9	αα Iα0α Iα0α0α Iα Iα Iα0α Iβ	
3	β Iα0α Iα0α0α Iα Iα Iα0α Iβ	
2	α Iβα0α Iα0α0α Iα Iα Iα0α Iβ	
7	α Iβ0α Iα0α0α Iα Iα Iα0α Iβ	
2	α Iα0βα Iα0α0α Iα Iα Iα0α Iβ	
7	α Iα0β Iα0α0α Iα Iα Iα0α Iβ	
2	α Iα0α Iβα0α0α Iα Iα Iα0α Iβ	
7	α Iα0α Iβ0α0α Iα Iα Iα0α Iβ	
2	α Iα0α Iα0βα0α Iα Iα Iα0α Iβ	
7	α Iα0α Iα0β0α Iα Iα Iα0α Iβ	
2	α Iα0α Iα0α0βα Iα Iα Iα0α Iβ	
7	α Iα0α Iα0α0β Iα Iα Iα0α Iβ	
2	α Iα0α Iα0α0α Iβα Iα Iα0α Iβ	
7	α Iα0α Iα0α0α Iβ Iα Iα0α Iβ	
2	α Iα0α Iα0α0α Iα Iβα Iα0α Iβ	
7	α Iα0α Iα0α0α Iα Iβ Iα0α Iβ	
2	α Iα0α Iα0α0α Iα Iα Iβα0α Iβ	
7	α Iα0α Iα0α0α Iα Iα Iβ0α Iβ	
2	α Iα0α Iα0α0α Iα Iα Iα0βα Iβ	
7	α Iα0α Iα0α0α Iα Iα Iα0β Iβ	
2	α Iα0α Iα0α0α Iα Iα Iα0α Iββ	
4	α Iα0α Iα0α0α Iα Iα Iα0α IΓ	
0	α Iα0α Iα0α0α Iα Iα Iα IΓ0α	
0	α Iα0α Iα0α0α Iα Iα IΓ Iα0α	
0	α Iα0α Iα0α0α Iα IΓ Iα Iα0α	

0	α1α0α1α0α0α1Γ1α1α1α0α	
0	α1α0α1α0α1Γ0α1α1α1α0α	
0	α1α0α1α1Γ0α0α1α1α1α0α	
0	α1α0α1Γ1α0α0α1α1α1α0α	
0	α1α1Γ0α1α0α0α1α1α1α0α	
0	α1Γ1α0α1α0α0α1α1α1α0α	
1	1Γα1α0α1α0α0α1α1α1α0α	
5	1Γ1α0α1α0α0α1α1α1α0α	
6	11Γα0α1α0α0α1α1α1α0α	
5	11Γ0α1α0α0α1α1α1α0α	
6	110Γα1α0α0α1α1α1α0α	
5	110Γ1α0α0α1α1α1α0α	
6	1101Γα0α0α1α1α1α0α	
5	1101Γ0α0α1α1α1α0α	
6	11010Γα0α1α1α1α0α	
5	11010Γ0α1α1α1α0α	
6	110100Γα1α1α1α0α	
5	110100Γ1α1α1α0α	
6	1101001Γα1α1α0α	
5	1101001Γ1α1α0α	
6	11010011Γα1α0α	
5	11010011Γ1α0α	
6	110100111Γα0α	
5	110100111Γ0α	
6	1101001110Γα	
5	1101001110Γ	
8	1101001110	← [Second cyclic shift is over after 130 elementary transformations]

4.2 HOW TO REALIZE A CONSTRUCTIVE SIGNAL PROCESSING OPERATION ?

Here, the term constructive signal processing refers to the symbolic processing of signals by normal algorithms, irrespective of the fact whether sample values of signals are expressed as numbers or words.

DEFINITION 4.2.1

By a *constructive signal processing system*, we mean an ordered pair $\langle \mathfrak{N}, X \rangle$, where, $\mathfrak{N}$ represents a normal algorithm and X is a subset of $(\mathcal{A} \cup \mathfrak{D})^*$ that is recognized by $\mathfrak{N}$. $\mathcal{A}$ is a basic alphabet in which signals are represented and $\mathfrak{D}$ is an alphabet of delimiters.

Now, we outline a general method for constructing normal algorithms for implementing binary operations between numerical data sequences.

Let us consider two numerical data sequences p(n) and q(n) of M and N data samples respectively. Let r(n) be the result of an arithmetic operation, say · between p(n) and q(n), that is, r(n) = p(n)·q(n). Our interest is to carry out this operation symbolically, by means of a normal algorithm. We proceed as follows.

Firstly, we code the data samples of p(n) and q(n) as words from a suitable alphabet $\mathcal{A}$ so that the data sequences p(n) and q(n) are expressed respectively as $P\uparrow d_p = P_1 d_p P_2 d_p \ldots P_i d_p \ldots d_p P_M$, $1 \leq i \leq M$ and $Q\uparrow d_q = Q_1 d_q Q_2 d_q \ldots Q_j d_q \ldots d_q Q_N$, $1 \leq j \leq N$ where d_p and d_q are two different delimiters from the alphabet $\mathfrak{D}$. Next, we express these coded strings as a ✱-pair $(P\uparrow d_p)✱(Q\uparrow d_q)$, where the symbol ✱ is a delimiter from the alphabet $\mathfrak{D}$.

Now, the ✱-pair is to be manipulated in such a way that the resulting string, say $R\uparrow d_r$, corresponds to the output r(n). A normal algorithm meant for this purpose would cause the following sequence of string manipulations.

Let $d_k(P\uparrow d_p)✱(Q\uparrow d_q) \doteq d_k P_1 d_p \ldots d_p P_M ✱ Q_1 d_q \ldots d_q Q_N$ where, d_k is another delimiter. d_k is used to identify P_1 at any stage of the manipulation. Firstly, the ✱-pair pattern consisting of the right most factor P_M of $P\uparrow d_p$ and the left most factor Q_1 of $Q\uparrow d_q$ is substituted by its value $P_M ✱ Q_1 R_{M1}$. R_{M1} is the result of a suitable manipulation of P_M and Q_1 , and it corresponds to the value of the intended operation between the last sample of the first data sequence p(n) and the first sample of the second data sequence q(n). Now P_M is shifted to the left end such that the resulting string would be of the form:

$$P_M d_k P_1 d_p \ldots P_{M-1} d_p ✱ Q_1 R_{M1} d_q \ldots d_q Q_N.$$

By assigning a value $d_p ✱ Q_1 d_c$ to the pattern $d_p ✱ Q_1$, a delimiter d_c is injected in the first d_q-term, which is used here in order to identify R_{M1} at any stage of the manipulation. Now, d_p is shifted to the left end such that the resulting string would be of the form:

$$d_p P_M d_k P_1 d_p \ldots P_{M-1} ✱ Q_1 d_c R_{M1} d_q \ldots d_q Q_N.$$

The above procedure is repeated for the $*$-pair pattern $P_{M-1}*Q_1$ such that the resulting string would be of the form :

$$d_pP_{M-1}d_pP_Md_kP_1d_p...P_{M-2}*Q_1d_cR_{(M-1)1}d_cR_{M1}d_q...d_qQ_N$$

We shall repeat this procedure till we arrive at a string which would be of the form :

$$P_1d_p...d_pP_Md_k*Q_1R_{11}d_cR_{21}d_c...d_cR_{M1}d_qQ_2d_q...d_qQ_N.$$

Now, we shall make use of the delimiter d_k in two ways : (i) d_k is used as a type-3 annihilator such that Q_1 is erased by assigning the value d_k* to the pattern d_k*Q_1 and (ii) d_k is allowed to inject a right shift operator δ into the right $*$-term when either a value $d_k*\delta d_c$ is assigned to the pattern d_k*d_c , or a value $d_k*\delta R_{ij}$, $1 \leq i \leq M$, $1 \leq j \leq N$ is assigned to the pattern d_k*R_{ij} or a value $d_k*\delta d_r$ is assigned to the pattern d_k*d_q. The moment δ is introduced, R_{11} , d_c and d_r will be shifted to the right. This is repeated till all the M generated values are shifted to the right end. Now, the delimiter d_k is shifted to the left such that the resulting string would be of the form :

$$d_kP_1d_p...d_pP_M*Q_2d_q...d_qQ_Nd_r\delta R_{M1}\delta d_c\delta R_{(M-1)1}...\delta d_c\delta R_{11}$$

This procedure is repeated for all the remaining (N-1) d_q-terms of $Q\uparrow d_q$ after which $P\uparrow d_p$ together with the delimiters d_k , $*$ and the right shift operator δ are erased by means of a type-3 and a type-2 annihilation formulas. The resulting string $R\uparrow d_r$ corresponds to the result of the operation between the two coded strings of p(n) and q(n) and it would be of the form :

$$R_{MN}d_cR_{(M-1)N}d_c...d_cR_{1N}d_rR_{M(N-1)}d_cR_{(M-1)(N-1)}d_c...d_r...R_{M1}d_cR_{(M-1)1}d_c...d_cR_{11}.$$

Now, the values of R_{ij} ; $1 \leq i \leq M$; $1 \leq j \leq N$ are decoded such that the output r(n) due to the operation $*$ between p(n) and q(n) , is obtained.

Note that the operation $\cdot$ between p(n) and q(n) is carried out by manipulating the corresponding coded $*$-pair with the help of left shifing, right shifting, transposition, annihilation and pattern matching substitution formulas.

4.3 LINEAR CONVOLUTION OF NONNEGATIVE INTEGER SEQUENCES BY A CONSTRUCTIVE SYSTEM $\mathfrak{R}^{con}$

In 4.2, a general method as to how one can construct a normal algorithmic signal processing system was outlined. In what follows, we provide a specific example of one way of constructing a normal algorithmic system, $\mathfrak{R}^{con}$, that carries out the operation of linear convolution of nonnegative integer sequences of arbitrary lengths. We shall make use of the string manipulating techniques of section 3 in obtaining our constructive system $\mathfrak{R}^{con}$.

Let us consider two discrete time signals x(n) and y(n) of samples M and N respectively, which are to be convolved with the help of normal algorithms. Let the convolved output be z(n) of M+N-1 samples. Now, the required system $\mathfrak{R}^{con}$ is constructed as per the following steps:

STEP 1

We need the following alphabets for the construction of $\mathfrak{R}^{con}$:

$\mathcal{A}_0$ = {0 |}; $\mathfrak{D}_1$ = { , } ; $\mathfrak{D}_2$ = {□} ; $\mathfrak{D}_3$ = {*}

$\mathcal{E}$ = { a b α β Γ δ θ σ ω ν }.

$\mathfrak{D} = \mathfrak{D}_1 \cup \mathfrak{D}_2 \cup \mathfrak{D}_3$; $\mathcal{A}_{11} = \mathcal{A}_0 \cup \mathfrak{D}_1$; $\mathcal{A}_{12} = \mathcal{A}_0 \cup \mathfrak{D}_2$; $\mathcal{A}_{13} = \mathcal{A}_{11} \cup \mathcal{A}_{12}$;

A_0 = {0 1 2 3 4 5 6 7 8 9}. $\mathcal{A}_{14} = \mathcal{A}_{13} \cup \mathfrak{D}_3$;

Further, we consider the sets N = {x | x is a natural number in the traditional sense} and S = {x | x is a nonnegative integer sequence}.

We also need the following subsets of certain free monoids [Table 4.3.1] for the construction of $\mathfrak{R}^{con}$:

Table 4.3.1: Subsets recognized by $\mathfrak{R}^{con}$

Sl.No	Free monoids	Subsets	sample elements of the subsets
1	A_0^*	N = {x | x is a word (natural number) from A_0.}	4
2	$(A_0 \cup \mathfrak{D}_1)^*$	S = {x | x is a nonnegative integer sequence.}	2,4,6,8,12
3	$\mathcal{A}_0^*$	X_0= {x | x=0 or I^i ; i≥1}	IIII
4	$\mathcal{A}_{11}^*$	X_{11}={x | x=P or P↑, ; P∈X_0}	I,II,III
5	$\mathcal{A}_{12}^*$	X_{12}={x | x=P or P↑□ ; P∈X_0}	II□III□IIII
6	$\mathcal{A}_{13}^*$	X'_{13}={x | x∈X_{11} or X_{12}}	I,II,III or II□III□IIII
7	$\mathcal{A}_{13}^*$	X''_{13}={x | x=P↑□ ; P∈X_{11}}	I,II,III□II,III□II,I
8	$\mathcal{A}_{14}^*$	X_{14} = {x | x=□P*Q ; P∈X_{11}; Q∈X_{12} or X''_{13}	□I,II,III*II□III□IIII or □I,II,I*II,III□I,II,III

STEP 2

Firstly, we code the data samples of x(n) and y(n) from $\mathbb{N}$ over the alphabet $\mathcal{A}_0$ in the following manner: The number 0 is represented as the string 0 and the number 1 is represented as the string I from the alphabet $\mathcal{A}_0$. The number 2 is represented as the string II and the number 3 is coded as the string III . In this fashion, any non-negative integer can be coded in the alphabet $\mathcal{A}_0$. Since convolution is a binary operation, we express the coded strings of x(n) and y(n) as a *-pair over the union of the alphabets $\mathcal{A}_0$, $\mathfrak{D}_1$, $\mathfrak{D}_2$ and $\mathfrak{D}_3$ so that the *-pair is of the form □X↑,*Y↑□ ≐ □X_1,X_2,...,X_i,...,X_M*Y_1□Y_2□ ... □Y_j□ ... □Y_N where X_i , $(1 \leq i \leq M)$ and Y_j , $(1 \leq j \leq N)$ are coded strings from $\mathcal{A}_0$ corresponding to the data samples of x(n) and y(n) respectively. We require three normal algorithms in order to obtain the above *-pair: (i) $\mathcal{N}^{\uparrow,}$ that transforms x(n) into X↑, (ii) $\mathcal{N}^{\uparrow\square}$ that transforms y(n)

into Y↑□ and (iii) $\mathcal{N}^{✱}$ that forms the ✱-pair □X↑,✱Y↑□ . For brevity, we shall call the string □X↑,✱Y↑□ *Preconvolution String*.

STEP 3

Convolution is basically a multiplication operation. Firstly, let us consider the following normal algorithm $\mathcal{N}^{mul}$ which allows one way of carrying out the multiplication operation between two coded strings of nonnegative integers that are expressed as a ✱-pair. $\mathcal{N}^{mul}$ has been constructed over $\mathcal{A}_0 \cup \mathcal{D}_3 \cup 8$.

<table>
<tr><td>$\mathcal{N}^{mul}$:</td><td>Substitution formulas</td><td>Formula number</td></tr>
<tr><td></td><td>a| ⟶ |ba</td><td>(0)</td></tr>
<tr><td></td><td>b| ⟶ |b</td><td>(1)</td></tr>
<tr><td></td><td>□✱| ⟶ □✱</td><td>(2)</td></tr>
<tr><td></td><td>|✱□ ⟶ ✱□</td><td>(3)</td></tr>
<tr><td></td><td>|✱| ⟶ ✱a|</td><td>(4)</td></tr>
<tr><td></td><td>✱| ⟶ ✱</td><td>(5)</td></tr>
<tr><td></td><td>a ⟶</td><td>(6)</td></tr>
<tr><td></td><td>✱b ⟶ |✱</td><td>(7)</td></tr>
<tr><td></td><td>✱ ⟶·</td><td>(8)</td></tr>
</table>

For example let us consider a ✱-pair |||✱|| of strings from $\mathcal{A}_0$ and apply $\mathcal{N}^{mul}$ to it.

<table>
<tr><td>$\mathcal{N}^{mul}$:</td><td>Formula number</td><td>Elementary transformations</td></tr>
<tr><td></td><td>-</td><td>|||✱|| ⟵ (Input String)</td></tr>
<tr><td></td><td>(4)</td><td>||✱a||</td></tr>
<tr><td></td><td>(0)</td><td>||✱|ba|</td></tr>
<tr><td></td><td>(0)</td><td>||✱|b|ba</td></tr>
<tr><td></td><td>(1)</td><td>||✱||bba</td></tr>
<tr><td></td><td>(4)</td><td>|✱a||bba</td></tr>
<tr><td></td><td>(0)</td><td>|✱|ba|bba</td></tr>
<tr><td></td><td>(0)</td><td>|✱|b|babba</td></tr>
<tr><td></td><td>(1)</td><td>|✱||bbabba</td></tr>
<tr><td></td><td>(4)</td><td>✱a||bbabba</td></tr>
<tr><td></td><td>(0)</td><td>✱|ba|bbabba</td></tr>
<tr><td></td><td>(0)</td><td>✱|b|babbabba</td></tr>
<tr><td></td><td>(1)</td><td>✱||bbabbabba</td></tr>
<tr><td></td><td>(5)</td><td>✱|bbabbabba</td></tr>
<tr><td></td><td>(5)</td><td>✱bbabbabba</td></tr>
<tr><td></td><td>(6)</td><td>✱bbbbabba</td></tr>
</table>

(6)	$\ast bbbbbba$	
(6)	$\ast bbbbbb$	
(7)	$\mid\ast bbbbb$	
(7)	$\mid\mid\ast bbbb$	
(7)	$\mid\mid\mid\ast bbb$	
(7)	$\mid\mid\mid\mid\ast bb$	
(7)	$\mid\mid\mid\mid\mid\ast b$	
(7)	$\mid\mid\mid\mid\mid\mid\ast$	
(8)	$\mid\mid\mid\mid\mid\mid$	← (Output String)

The string IIIIII corresponds to the number 6 which is the product of 3 and 2 .

Now, the multiplication operation can be carried out in the ✱-pair :

$$\square X\uparrow,\ast Y\uparrow\square \doteqdot \square X_1,X_2,\ldots,X_i,\ldots,X_M\ast Y_1\square Y_2\square \ldots \square Y_j\square \ldots \square Y_N$$

by means of a normal algorithm which makes use of the technique of $\mathcal{N}^{mul}$.

$\mathcal{N}^{PP}$ is one such normal algorithm constructed over $\mathcal{A}_0\cup\mathfrak{D}\cup\mathfrak{B}$. It contains certain pattern substitutions and certain already known algorithms. The scheme of $\mathcal{N}^{PP}$ is given below : [Note: (i) $\mu, \xi \in \mathcal{A}_0$. (ii) The acronyms TPT stands for *Transposition Type*, AT for *Annihilation Type*, PTS for *Pattern Substitution* and SPL for *Special* type. A special type formula is constructed by making use of known techniques as per the need. (iii) LINCON is the FORTRAN program for implementing $\mathfrak{R}^{con}$ [Appendix A.2]]

Sl.No.	Substitution Formulas	Formula number as labelled in program (LINCON)	Type of the formula	Remarks
(00)	$\mu\alpha\xi\Gamma \longrightarrow \xi\Gamma\mu\alpha$	(00)	TPT-4	The formulas (00) to (12), (59) and (60) form a modified scheme of $\mathcal{N}^{ccs}$.
(01)	$\square\alpha\xi\Gamma \longrightarrow \xi\Gamma\square\alpha$	(01)	TPT-4	
(02)	$\mu\alpha\square\Gamma \longrightarrow \square\Gamma\mu\alpha$	(02)	TPT-4	
(03)	$\alpha\square\Gamma \longrightarrow \square\Gamma\alpha$	(03)	TPT-4	
(04)	$\alpha\xi\Gamma \longrightarrow \xi\Gamma\alpha$	(04)	TPT-4	
(05)	$\beta\mu \longrightarrow \alpha\mu\beta$	(05)	SPL	
(06)	$\alpha\alpha \longrightarrow \beta$	(06)	SPL	
(07)	$\beta\beta \longrightarrow \Gamma$	(07)	SPL	
(08)	$\Gamma\alpha \longrightarrow \Gamma$	(08)	AT-2	
(09)	$\Gamma\mu \longrightarrow \mu\Gamma$	(09)	TPT-3	
(10)	$\Gamma\square \longrightarrow \square\Gamma$	(10)	TPT-3	
(11)	$\beta\alpha \longrightarrow \beta$	(11)	AT-2	
(12)	$\beta\square \longrightarrow \alpha\square\beta$	(12)	SPL	
(13)	$\mu\ast\nu \longrightarrow \ast\nu$	(13)	AT-3	Formulas (13),(14) and (15) form the scheme of $\mathcal{N}^{LEX}$
(14)	$\square\ast\nu \longrightarrow \ast\nu$	(14)	AT-3	
(15)	$\ast\nu \longrightarrow \nu$	(15)	AT-2	

(16)	$\nu\delta \longrightarrow \nu$	(16)	AT-2	
(17)	$\nu\sigma \longrightarrow \nu$	(17)	AT-2	
(18)	$\nu\vert \longrightarrow \vert\nu$	(18)	composition of TPT-3 and letter substitution	
(19)	$\nu\Box \longrightarrow \Box\nu$	(19)	TPT-3	
(20)	$\nu\mu \longrightarrow \mu\nu$	(20)	TPT-3	
(21)	$\delta\sigma \longrightarrow \delta$	(21)	AT-2	
(22)	$\Gamma\beta \longrightarrow \Gamma$	(22)	AT-2	
(23)	$\Box\ast\vert \longrightarrow \Box\beta\ast\Box\vert$	(23)	PTS	
(24)	$\Box\ast\Box \longrightarrow \Box\ast a\Box$	(24)	PTS	
(25)	$,\ast \longrightarrow ,\beta\ast,$	(25)	PTS	
(26)	$\vert\ast\Box \longrightarrow \vert\ast a\Box$	(26)	PTS	
(27)	$a\vert \longrightarrow \vert ba$	(27)	SPL	
(28)	$\delta b\mu \longrightarrow \mu\delta b$	(28)	TPT-4	
(29)	$a\Box a \longrightarrow aa\Box$	(29)	TPT-3	
(30)	$a\Box\Box \longrightarrow a\Box$	(30)	AT-1	
(31)	$aa \longrightarrow a$	(31)	AT-1	
(32)	$b\vert \longrightarrow \vert b$	(32)	TPT-3	
(33)	$\delta b\Box \longrightarrow \Box\delta b$	(33)	TPT-4	Formulas (28)
(34)	$\delta ba \longrightarrow a\delta b$	(34)	TPT-4	(33) to (37) ,
(35)	$\delta bb \longrightarrow b\delta b$	(35)	TPT-4	(39) to (44) form
(36)	$\delta b\sigma \longrightarrow \sigma\delta b$	(36)	TPT-4	the scheme of
(37)	$\delta\mu\xi \longrightarrow \xi\delta\mu$	(37)	TPT-4	$\mathfrak{N}^{RS}$.
(38)	$\Box\ast\delta\Box \longrightarrow \Box\ast\Box$	(38)	AT-3	
(39)	$\delta\mu\Box \longrightarrow \Box\delta\mu$	(39)	TPT-4	
(40)	$\delta\mu\sigma \longrightarrow \sigma\delta\mu$	(40)	TPT-4	
(41)	$\delta\mu a \longrightarrow a\delta\mu$	(41)	TPT-4	
(42)	$\delta\mu b \longrightarrow b\delta\mu$	(42)	TPT-4	
(43)	$\delta\Box\mu \longrightarrow \mu\delta\Box$	(43)	TPT-4	
(44)	$\delta\Box\Box \longrightarrow \Box\delta\Box$	(44)	TPT-4	
(45)	$\sigma\mu\xi \longrightarrow \xi\sigma\mu$	(45)	TPT-4	
(46)	$\Box\ast\vert \longrightarrow \Box\ast$	(46)	AT-3	Formulas (46),
(47)	$\Box\ast\Box \longrightarrow \Box\ast$	(47)	AT-3	(47),(49),(50)
(48)	$\Box\ast ba \longrightarrow \Box\ast\delta b$	(48)	PTS	and (51) form
(49)	$\Box\ast b \longrightarrow \Box\ast\delta\vert$	(49)	composition of AT-3 and letter substitution	the scheme of $\mathfrak{N}^{REX}$.
(50)	$\Box\ast a \longrightarrow \Box\ast\delta$	(50)	composition of AT-3 and PTS	
(51)	$\Box\ast\sigma \longrightarrow \Box\ast\delta$	(51)	composition of AT-3 and PTS	
(52)	$\Gamma\ast\Box \longrightarrow \ast\delta\Box$	(52)	composition of AT-3 and PTS	
(53)	$\Gamma\ast, \longrightarrow \ast\sigma,$	(53)	PTS	
(54)	$\Gamma\ast b \longrightarrow \ast\delta b$	(54)	PTS	
(55)	$\Gamma\ast\vert \longrightarrow \ast a\vert$	(55)	PTS	
(56)	$\Gamma\ast\Box \longrightarrow \ast\sigma\Box$	(56)	PTS	
(57)	$\Gamma\ast a\Box \longrightarrow \ast\Box a\Box$	(57)	PTS	

(58)	$\Gamma * \delta \longrightarrow * \nu \delta$	(58)	composition of AT-3 and PTS
(59)	$\nu \longrightarrow \cdot$	(59)	AT-1
(60)	$\longrightarrow \alpha$	(60)	SPL

$\mathcal{N}^{PP}$ transforms the pre-convolution string □X↑,*Y↑□ into another string of the form $(P\uparrow\square)^{-1}$ which is the inverse of P↑□ [Definition 2.3.4]. The string P↑□ ≐ $P_{11},P_{21},\ldots,P_{M1}\square P_{12},P_{22},\ldots,P_{M2}\square\ldots\square P_{1N},P_{2N},\ldots,P_{MN}$ is called the *partial product string,* and every □-term of P↑□ corresponds to the *partial product component* of the multiplication of y(n) by x(n).

For example, let us consider two data sequences x(n) = 1, 0 and y(n) = 2, 1 and apply $\mathcal{N}^{PP}$ to the corresponding pre-convolution string □I,0*II□I. After 269 elementary transformations we would obtain the string 0,I00,II which is the inverse of the partial product string P↑□. [NOTE: We have made use of the program LINCON in obtaining the required string]

FORMULA NUMBER	ELEMENTARY TRANSFORMATIONS	Remarks
	The pre-convolution string is:	Data sequences to be convolved: x(n)=1,0 & y(n)=2,1
	□I,0*II□I	
23	□I,0β*0II□I ------------------------>	Product of 0 and 2 is obtained in the coded form as 0
60	α□I,0β*0II□I	
60	αα□I,0β*0II□I	
6	β□I,0β*0II□I	
12	α□βI,0β*0II□I	
5	α□αIβ,0β*0II□I	
5	α□αIα,β0β*0II□I	
5	α□αIα,α0ββ*0II□I	
7	α□αIα,α0Γ*0II□I	
0	α□αIα0Γ,α*0II□I	
0	α□α0ΓIα,α*0II□I	
0	α0Γ□αIα,α*0II□I	
4	0Γα□αIα,α*0II□I	
8	0Γ□αIα,α*0II□I	
10	0□ΓαIα,α*0II□I	
8	0□ΓIα,α*0II□I	
9	0□IΓα,α*0II□I	
8	0□IΓ,α*0II□I	
9	0□I,Γα*0II□I	

```
 8  0□I,Γ✱0II□I   -------------------------->   The coded string of 0 left shifted
56  0□I,✱σ0II□I
25  0□I,β✱,σ0II□I
45  0□I,β✱,Iσ0I□I
45  0□I,β✱,IIσ0□I
60  α0□I,β✱,IIσ0□I
60  αα0□I,β✱,IIσ0□I
 6  β0□I,β✱,IIσ0□I
 5  α0β□I,β✱,IIσ0□I
12  α0α□βI,β✱,IIσ0□I
 5  α0α□αIβ,β✱,IIσ0□I
 5  α0α□αIα,ββ✱,IIσ0□I
 7  α0α□αIα,Γ✱,IIσ0□I
 0  α0α□α,ΓIα✱,IIσ0□I
 0  α0α,Γ□αIα✱,IIσ0□I
 0  α,Γ0α□αIα✱,IIσ0□I
 4  ,Γα0α□αIα✱,IIσ0□I
 8  ,Γ0α□αIα✱,IIσ0□I
 9  ,0Γα□αIα✱,IIσ0□I
 8  ,0Γ□αIα✱,IIσ0□I
10  ,0□ΓαIα✱,IIσ0□I
 8  ,0□ΓIα✱,IIσ0□I
 9  ,0□IΓα✱,IIσ0□I
 8  ,0□IΓ✱,IIσ0□I
53  ,0□I✱σ,IIσ0□I
45  ,0□I✱Iσ,Iσ0□I
45  ,0□I✱IIσ,σ0□I
60  α,0□I✱IIσ,σ0□I
60  αα,0□I✱IIσ,σ0□I
 6  β,0□I✱IIσ,σ0□I
 5  α,β0□I✱IIσ,σ0□I
 5  α,α0β□I✱IIσ,σ0□I
12  α,α0α□βI✱IIσ,σ0□I
 5  α,α0α□αIβ✱IIσ,σ0□I
60  αα,α0α□αIβ✱IIσ,σ0□I
 6  β,α0α□αIβ✱IIσ,σ0□I
 5  α,βα0α□αIβ✱IIσ,σ0□I
11  α,β0α□αIβ✱IIσ,σ0□I
 5  α,α0βα□αIβ✱IIσ,σ0□I
11  α,α0β□αIβ✱IIσ,σ0□I
12  α,α0α□βαIβ✱IIσ,σ0□I
11  α,α0α□βIβ✱IIσ,σ0□I
 5  α,α0α□αIββ✱IIσ,σ0□I
 7  α,α0α□αIΓ✱IIσ,σ0□I
 0  α,α0αIΓ□α✱IIσ,σ0□I
 0  α,αIΓ0α□α✱IIσ,σ0□I
 0  αIΓ,α0α□α✱IIσ,σ0□I
 4  IΓα,α0α□α✱IIσ,σ0□I
 8  IΓ,α0α□α✱IIσ,σ0□I
 9  I,Γα0α□α✱IIσ,σ0□I
 8  I,Γ0α□α✱IIσ,σ0□I
 9  I,0Γα□α✱IIσ,σ0□I
 8  I,0Γ□α✱IIσ,σ0□I
10  I,0□Γα✱IIσ,σ0□I
```

```
 8  l,0□Γ✱llσ,σ0□l
55  l,0□✱allσ,σ0□l
27  l,0□✱lbalσ,σ0□l
27  l,0□✱lblbaσ,σ0□l
32  l,0□✱llbbaσ,σ0□l
46  l,0□✱lbbaσ,σ0□l
46  l,0□✱bbaσ,σ0□l   -------------------->   Product of 1 and 2 is obtained
49  l,0□✱δbbaσ,σ0□l                         in the coded form as  bb.
35  l,0□✱bδbaσ,σ0□l
34  l,0□✱baδbσ,σ0□l
36  l,0□✱baσδb,σ0□l
28  l,0□✱baσ,δbσ0□l
36  l,0□✱baσ,σδb0□l
28  l,0□✱baσ,σ0δb□l
33  l,0□✱baσ,σ0□δbl
28  l,0□✱baσ,σ0□lδb
48  l,0□✱δbσ,σ0□lδb
36  l,0□✱σδb,σ0□lδb
28  l,0□✱σ,δbσ0□lδb
36  l,0□✱σ,σδb0□lδb
28  l,0□✱σ,σ0δb□lδb
33  l,0□✱σ,σ0□δblδb
28  l,0□✱σ,σ0□lδbδb
51  l,0□✱δ,σ0□lδbδb
40  l,0□✱σδ,0□lδbδb
37  l,0□✱σ0δ,□lδbδb
39  l,0□✱σ0□δ,lδbδb
37  l,0□✱σ0□lδ,δbδb
51  l,0□✱δ0□lδ,δbδb
39  l,0□✱□δ0lδ,δbδb
37  l,0□✱□lδ0δ,δbδb   ------------------->   Coded string corresponding
60  αl,0□✱□lδ0δ,δbδb                        to the first partial product
60  ααl,0□✱□lδ0δ,δbδb                       component right shifted.
 6  βl,0□✱□lδ0δ,δbδb
 5  αlβ,0□✱□lδ0δ,δbδb
 5  αlα,β0□✱□lδ0δ,δbδb
 5  αlα,α0β□✱□lδ0δ,δbδb
12  αlα,α0α□β✱□lδ0δ,δbδb
60  ααlα,α0α□β✱□lδ0δ,δbδb
 6  βlα,α0α□β✱□lδ0δ,δbδb
 5  αlβα,α0α□β✱□lδ0δ,δbδb
11  αlβ,α0α□β✱□lδ0δ,δbδb
 5  αlα,βα0α□β✱□lδ0δ,δbδb
11  αlα,β0α□β✱□lδ0δ,δbδb
 5  αlα,α0βα□β✱□lδ0δ,δbδb
11  αlα,α0β□β✱□lδ0δ,δbδb
12  αlα,α0α□ββ✱□lδ0δ,δbδb
 7  αlα,α0α□Γ✱□lδ0δ,δbδb
 0  αlα,α□Γ0α✱□lδ0δ,δbδb
 0  αlα□Γ,α0α✱□lδ0δ,δbδb
 0  α□Γlα,α0α✱□lδ0δ,δbδb
 3  □Γαlα,α0α✱□lδ0δ,δbδb
 8  □Γlα,α0α✱□lδ0δ,δbδb
 9  □lΓα,α0α✱□lδ0δ,δbδb
```

```
 5   α,βO□|*|σ,σOδ□δOδ,δbδb
 5   α,αOβ□|*|σ,σOδ□δOδ,δbδb
12   α,αOα□β|*|σ,σOδ□δOδ,δbδb
 5   α,αOα□α|β*|σ,σOδ□δOδ,δbδb
60   αα,αOα□α|β*|σ,σOδ□δOδ,δbδb
 6   β,αOα□α|β*|σ,σOδ□δOδ,δbδb
 5   α,βαOα□α|β*|σ,σOδ□δOδ,δbδb
11   α,βOα□α|β*|σ,σOδ□δOδ,δbδb
 5   α,αOβα□α|β*|σ,σOδ□δOδ,δbδb
11   α,αOβ□α|β*|σ,σOδ□δOδ,δbδb
12   α,αOα□βα|β*|σ,σOδ□δOδ,δbδb
11   α,αOα□β|β*|σ,σOδ□δOδ,δbδb
 5   α,αOα□α|ββ*|σ,σOδ□δOδ,δbδb
 7   α,αOα□α|Γ*|σ,σOδ□δOδ,δbδb
 0   α,αOα|Γ□α*|σ,σOδ□δOδ,δbδb
 0   α,α|ΓOα□α*|σ,σOδ□δOδ,δbδb
 0   α|Γ,αOα□α*|σ,σOδ□δOδ,δbδb
 4   |Γα,αOα□α*|σ,σOδ□δOδ,δbδb
 8   |Γ,αOα□α*|σ,σOδ□δOδ,δbδb
 9   |,ΓαOα□α*|σ,σOδ□δOδ,δbδb
 8   |,ΓOα□α*|σ,σOδ□δOδ,δbδb
 9   |,OΓα□α*|σ,σOδ□δOδ,δbδb
 8   |,OΓ□α*|σ,σOδ□δOδ,δbδb
10   |,O□Γα*|σ,σOδ□δOδ,δbδb
 8   |,O□Γ*|σ,σOδ□δOδ,δbδb
55   |,O□*a|σ,σOδ□δOδ,δbδb
27   |,O□*|baσ,σOδ□δOδ,δbδb
46   |,O□*baσ,σOδ□δOδ,δbδb      ---------->   Product of 1 and 1 is obtained
48   |,O□*δbσ,σOδ□δOδ,δbδb                    in the coded form as  b
36   |,O□*σδb,σOδ□δOδ,δbδb
28   |,O□*σ,δbσOδ□δOδ,δbδb
36   |,O□*σ,σδbOδ□δOδ,δbδb
28   |,O□*σ,σOδbδ□δOδ,δbδb
51   |,O□*δ,σOδbδ□δOδ,δbδb
40   |,O□*σδ,Oδbδ□δOδ,δbδb
37   |,O□*σOδ,δbδ□δOδ,δbδb
51   |,O□*δOδ,δbδ□δOδ,δbδb
60   α|,O□*δOδ,δbδ□δOδ,δbδb
60   αα|,O□*δOδ,δbδ□δOδ,δbδb
 6   β|,O□*δOδ,δbδ□δOδ,δbδb
 5   α|β,O□*δOδ,δbδ□δOδ,δbδb
 5   α|α,βO□*δOδ,δbδ□δOδ,δbδb
 5   α|α,αOβ□*δOδ,δbδ□δOδ,δbδb
12   α|α,αOα□β*δOδ,δbδ□δOδ,δbδb
60   αα|α,αOα□β*δOδ,δbδ□δOδ,δbδb
 6   β|α,αOα□β*δOδ,δbδ□δOδ,δbδb
 5   α|βα,αOα□β*δOδ,δbδ□δOδ,δbδb
11   α|β,αOα□β*δOδ,δbδ□δOδ,δbδb
 5   α|α,βαOα□β*δOδ,δbδ□δOδ,δbδb
11   α|α,βOα□β*δOδ,δbδ□δOδ,δbδb
 5   α|α,αOβα□β*δOδ,δbδ□δOδ,δbδb
11   α|α,αOβ□β*δOδ,δbδ□δOδ,δbδb
12   α|α,αOα□ββ*δOδ,δbδ□δOδ,δbδb
 7   α|α,αOα□Γ*δOδ,δbδ□δOδ,δbδb
```

```
 0  α|α,α□Γ0α*δ0δ,δbδ□δ0δ,δbδb
 0  α|α□Γ,α0α*δ0δ,δbδ□δ0δ,δbδb
 0  α□Γ|α,α0α*δ0δ,δbδ□δ0δ,δbδb
 3  □Γα|α,α0α*δ0δ,δbδ□δ0δ,δbδb
 8  □Γ|α,α0α*δ0δ,δbδ□δ0δ,δbδb
 9  □|Γα,α0α*δ0δ,δbδ□δ0δ,δbδb
 8  □|Γ,α0α*δ0δ,δbδ□δ0δ,δbδb
 9  □|,Γα0α*δ0δ,δbδ□δ0δ,δbδb
 8  □|,Γ0α*δ0δ,δbδ□δ0δ,δbδb
 9  □|,0Γα*δ0δ,δbδ□δ0δ,δbδb
 8  □|,0Γ*δ0δ,δbδ□δ0δ,δbδb
58  □|,0*νδ0δ,δbδ□δ0δ,δbδb
13  □|,*νδ0δ,δbδ□δ0δ,δbδb
13  □|*νδ0δ,δbδ□δ0δ,δbδb
13  □*νδ0δ,δbδ□δ0δ,δbδb
14  *νδ0δ,δbδ□δ0δ,δbδb
15  νδ0δ,δbδ□δ0δ,δbδb   -------------->    Left excision of the *-pair
16  ν0δ,δbδ□δ0δ,δbδb                        has been carried out
20  0νδ,δbδ□δ0δ,δbδb
16  0ν,δbδ□δ0δ,δbδb
20  0,νδbδ□δ0δ,δbδb
16  0,νbδ□δ0δ,δbδb
18  0,|νδ□δ0δ,δbδb
16  0,|ν□δ0δ,δbδb
19  0,|□νδ0δ,δbδb
16  0,|□ν0δ,δbδb
20  0,|□0νδ,δbδb
16  0,|□0ν,δbδb
20  0,|□0,νδbδb
16  0,|□0,νbδb
18  0,|□0,|νδb
16  0,|□0,|νb
18  0,|□0,||ν
```

THE INVERSE OF THE PARTIAL PRODUCT STRING IS:

```
59  0,|□0,||
```

The no. of elementary transformations is : 269

The complexity of symbol manipulation in terms of elementary transformations for the multiplication of any two nonnegative integers say i and j by $\mathcal{N}^{PP}$, is determined by the following rules :

[NOTE : $\mathcal{N}^{PP}$ has to be applied only to the *-pair I*J where I and J are the coded strings from $\mathcal{A}_0$ corresponding to the numbers i and j . Let t_{ij} be the number of elementary transformations of I*J after which the string coresponding to the product of i and j is obtained.]

RULE 1 : $t_{00} = 43$

RULE 2 : $t_{01} = 39$

RULE 3 : $t_{10} = 43$ [NOTE : $t_{01} \neq t_{10}$]

RULE 4 : For all $i > 1$, $t_{i0} = t_{(i-1)0} + (12i+16)$

RULE 5 : For all $j > 1$, $t_{0j} = t_{0(j-1)} + 2$

RULE 6 : For all $i > 0$, $t_{i1} = t_{i0} + i(i-1)$

RULE 7 : For all $i \geq 1$ and $j > 1$, $t_{ij} = t_{i(j-1)} + ij(i+1) + (3i+1)$

The following table contains cert. in values of t_{ij} , ($i, j \leq 8$).

Table 4.3.2: Complexity of symbol manipulation due to $\mathcal{N}^{PP}$ in multiplying two nonnegative integers

i \ j	0	1	2	3	4	5	6	7	8	→
0	43	39	41	43	45	47	49	51	53	→
1	43	43	51	61	73	87	103	121	141	→
2	83	85	104	129	160	197	240	289	344	→
3	135	141	175	221	279	349	431	525	631	→
4	199	211	264	337	430	543	676	829	1002	→
5	275	295	371	477	613	779	975	1201	1457	→
6	363	393	496	641	828	1057	1328	1641	1996	→
7	463	505	639	829	1075	1377	1735	2149	2619	→
8	575	631	800	1041	1354	1739	2196	2725	3326	→
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	

STEP 4

As mentioned in STEP 3, the string $P\uparrow\square \doteq P_{11}, P_{21}, \ldots\ldots, P_{M1}\square P_{12}, P_{22}, \ldots\ldots\ldots\ldots, P_{M2}\square \ldots\ldots\ldots\ldots\ldots \square P_{1N}, P_{2N}, \ldots\ldots, P_{MN}$ is called the partial product string, and every $\square$-term of $P\uparrow\square$ corresponds to the partial product component of the multiplication of y(n) by x(n). In the traditional signal processing, the linear or aperiodic convolution of x(n) of M samples and y(n) of N samples is computed by

$$z(n) = \sum_{k=0}^{M+N-1} y(k)x(n-k)$$

such that the convolved output z(n) consists of M+N-1 samples. Let us take for example, x(n) = 1,2,3 and y(n) = 2,3,4 and compute their convolution directly.

		y(n) 2	3	4
	1	2	3	4
x(n)	2	4	6	8
	3	6	9	12

Partial product components: ⟨ 2,4,6 ⟩, ⟨ 3,6,9 ⟩, ⟨ 4, 8 12 ⟩

The convolved output z(n) = 2, 7, 16, 17, 12 .

From the above computation, we observe that the first <u>one</u> element of the first component is taken as the first convolved output sample; <u>two</u> elements, 4 from the first component and 3 from the second component are added to get the second sample; <u>three</u> elements, 6 from the first component, 6 from the second component and 4 from the third component are added to get the third sample; <u>two</u> elements, 9 from the second component and 8 from the third component are added to get the fourth sample; and the remaining <u>one</u> element from the third component is taken as the fifth sample of the sequence z(n). Now, the ordered sequence of numbers that

are marked above with underline, can be seen to be of a palindrome type. Let us denote it by L. Then, L in this case is 1,2,3,2,1.

In general, three types of such palindrome sequences can be formed in the case of linear convolution of data sequences of M and N samples.

(i) 1,2,3,......................(n-1),n,(n-1),..............................,3,2,1 for M=N=n

(ii) 1,2,3,.....................,(M-1),M,M,M,................,M,(M-1),...,3,2,1 for M < N

N-M+1 times

(iii) 1,2,3, ... ,(N-1),N,N,N,.................,N,(N-1),...,3,2,1 for N < M

M-N+1 times

On the basis of this simple observation, we develop in what follows, a normal algorithm $\mathcal{N}^{CA}$ which chooses appropriate elements from $(P\uparrow D)^{-1}$ and add them so that a string $Z\uparrow, = Z_1,Z_2,...,Z_k,...,Z_{M+N-1}$ is obtained. Each ,-term of Z↑, corresponds to a sample of the convolved output z(n) of x(n) and y(n).

Two nonnegative integers, that is, words from $\mathcal{A}_0$ which are separated by a delimiter could be added just by removing the delimiter itself. For example, the representations of two integers 3 and 4 over the alphabet $\mathcal{A}_0$ can be added using the following scheme $\mathcal{N}^{+}_{\mathbb{N}}$ constructed over $\mathcal{A}_0 \cup \mathcal{D}_3$.

$\mathcal{N}^{+}_{\mathbb{N}}$:

	Formula number
$0* \longrightarrow$	(0)
$*0 \longrightarrow$	(1)
$* \longrightarrow$	(2)
$\longrightarrow\cdot$	(3)

$\mathcal{N}^{+}_{\mathbb{N}}$: $|||*|||| \vdash |||||| \vdash\cdot ||||||$

However, this algorithm fails in adding selected number representations in a string such as in the case of adding P_{11} , P_{12} and P_{13} in the string :

$$P_{11},P_{21},P_{31}\square P_{12},P_{22},P_{32}\square P_{13},P_{23},P_{33}$$

In such cases, the following prescription would be of use in carrying out

specific number of *Choose* and *Add* operations in arbitrarily long sequences similar to the one given above :

PRESCRIPTION FOR CHOOSE AND ADD OPERATIONS

(i) Let us consider a string P↑□ where each □-term is a ,-system of words P_{ij} from $\mathcal{A}_0$. Let us assume that there are ℓ number of □-terms in P↑□ and that our purpose is to choose the first ,-term of every □-term and add them.

(ii) Let L be the word from $\mathcal{A}_0$ corresponding to the number ℓ. Now we shall form the ✱-pair L✱P↑□ and apply the following scheme $\mathcal{N}^{ca}$ constructed over $\mathcal{A}_0 \cup \mathfrak{D} \cup \mathfrak{E}$:

$\mathcal{N}^{ca}$:

		Formula number
$\delta\mu\square \longrightarrow \square\delta\mu$	$(\mu \in \mathcal{A}_0)$	(00)
$\delta\mu\xi \longrightarrow \xi\delta\mu$	$(\xi \in \mathcal{A}_0)$	(01)
$\delta\mu, \longrightarrow ,\delta\mu$		(02)
$\vert{*}\mu\xi \longrightarrow \vert{*}\delta\mu\xi$		(03)
$\vert{*}\mu \longrightarrow {*}\delta\mu$		(04)
$\vert{*}, \longrightarrow {*},\nu$		(05)
${*}, \longrightarrow {*}$		(06)
$\nu\nu\square \longrightarrow \nu\square\nu$		(07)
$\nu\mu \longrightarrow \mu\nu$		(08)
$\nu, \longrightarrow ,\nu$		(09)
$\nu\square, \longrightarrow \square$		(10)
$\nu\square\mu \longrightarrow \nu\square\delta\mu$		(11)
$\delta \longrightarrow$		(12)
$\longrightarrow\cdot$		(13)

Now, let $P\uparrow\square \doteq P_{11},P_{21},\ldots,P_{M1}\square P_{12},P_{22},\ldots,P_{M2}\square\ldots\square P_{1N},P_{2N},\ldots,P_{MN}$; ℓ be the number of □-terms where, ℓ=M+N-1 and apply $\mathcal{N}^{ca}$ to the ✱-pair:

$$\underbrace{|||||\ldots\ldots\ldots|}_{\ell \text{ times}}{*}P_{11},P_{21},\ldots,P_{M1}\square P_{12},\ldots,P_{M2}\square\ldots\square P_{1N},\ldots,P_{MN}.$$

Due to the formulas (03) and (04) of $\mathcal{N}^{ca}$ the first ,-term P_{11} is shifted to the right, letter by letter, so that the resulting string would be of the form:

$$\underbrace{||||| \ldots\ldots\ldots\ldots |}_{\ell\text{-}1 \text{ times}} *, P_{21},\ldots,P_{M1}\Box P_{12},\ldots,P_{M2}\Box\ldots\Box P_{1N},\ldots,P_{MN}\delta P_{11}.$$

Now, the repetitive application of the formula (05) to the above string injects ℓ-1 number of choose operators ν in the right $*$-term which in turn get distributed to the remaining (ℓ-1) $\Box$-terms with the help of formulas (07), (08) and (09) of the scheme, as shown below :

$$*P_{21},\ldots,P_{M1}\nu\Box P_{12},\ldots,P_{M2}\nu\Box P_{13},\ldots\nu\Box\ldots\nu\Box P_{1N},\ldots,P_{MN}\ \delta P_{11}.$$

Now, with the help of formula (11), that is, $\nu\Box\mu \longrightarrow \mu\Box\delta\mu$ the first ,-term of each of the remaining (ℓ-1) $\Box$-terms is chosen one by one and shifted to right. After completing the shifting of all the chosen ,-terms , the shift operator δ is erased with the help of formula (12), and the required string of the following form $P_{21},\ldots,P_{M1}\Box P_{22},\ldots,P_{M2}\Box P_{23},\ldots\Box\ldots\Box P_{2N},\ldots,P_{MN}\ P_{1N}P_{1(N-1)}\ldots P_{12}P_{11}$ is obtained. Now, the factor $P_{1N}P_{1(N-1)}\ldots P_{12}P_{11}$ is the string corresponding to the addition of the first ,-term of all the $\Box$-terms of the string $P\uparrow\Box$.

A modified version of $\mathcal{N}^{ca}$ is the following scheme $\mathcal{N}^{CA}$ which carries out the required type of *Choose* and *Add* operations in an inverse partial product string.

SCHEME OF $\mathcal{N}^{CA}$:

Sl.No.	Substitution formulas	Formula number as labelled in Program (LINCON)	Type of the formula	Remarks
(00)	$\mu\alpha\xi\Gamma \longrightarrow \xi\Gamma\mu\alpha$	(61)	TPT-4	
(01)	$\Box\alpha\xi\Gamma \longrightarrow \xi\Gamma\Box\alpha$	(62)	TPT-4	
(02)	$\mu\alpha\Box\Gamma \longrightarrow \Box\Gamma\mu\alpha$	(63)	TPT-4	
(03)	$\alpha\Box\Gamma \longrightarrow \Box\Gamma\alpha$	(64)	TPT-4	

(04)	$\alpha\xi\Gamma \longrightarrow \xi\Gamma\alpha$	(65)	TPT-4
(05)	$\beta\mu \longrightarrow \alpha\mu\beta$	(66)	SPL
(06)	$\alpha\alpha \longrightarrow \beta$	(67)	SPL
(07)	$\beta\beta \longrightarrow \Gamma$	(68)	SPL
(08)	$\Gamma\alpha \longrightarrow \Gamma$	(69)	AT-2
(09)	$\Gamma\mu \longrightarrow \mu\Gamma$	(70)	TPT-3
(10)	$\Gamma\square \longrightarrow \square\Gamma$	(71)	TPT-3
(11)	$\beta\alpha \longrightarrow \beta$	(72)	AT-2
(12)	$\beta\square \longrightarrow \alpha\square\beta$	(73)	SPL
(13)	$\mu\ast\theta \longrightarrow \ast\theta$	(74)	AT-3
(14)	$\square\ast\theta \longrightarrow \ast\theta$	(75)	AT-3
(15)	$\ast\theta \longrightarrow \theta$	(76)	AT-2
(16)	$\theta\delta \longrightarrow \theta$	(77)	AT-2
(17)	$\theta\mu \longrightarrow \mu\theta$	(78)	TPT-3
(18)	$\delta\mu\square \longrightarrow \square\delta\mu$	(79)	TPT-4
(19)	$\delta\mu\xi \longrightarrow \xi\delta\mu$	(80)	TPT-4
(20)	$I\ast, \longrightarrow I\ast\omega$	(81)	PTS
(21)	$,\ast,\omega \longrightarrow ,\ast\omega$	(82)	PTS
(22)	$,\ast\omega \longrightarrow ,\ast\nu$	(83)	PTS
(23)	$\nu\omega \longrightarrow \omega\nu$	(84)	TPT-3
(24)	$\nu\nu\square \longrightarrow \nu\square\nu$	(85)	TPT-3
(25)	$,\ast\mu \longrightarrow ,\beta\ast,\mu$	(86)	PTS
(26)	$,, \longrightarrow ,$	(87)	SPL
(27)	$\square\square \longrightarrow \square$	(88)	AT-1
(28)	$I\square \longrightarrow I$	(89)	AT-1
(29)	$\square I \longrightarrow I$	(90)	AT-1
(30)	$\nu\mu \longrightarrow \mu\nu$	(91)	TPT-3

(31)	$\nu\square, \longrightarrow \square$	(92)	composition of two formulas of AT-1	otherwise known as "Excising formula"
(32)	$\nu\square\mu \longrightarrow \nu\square\delta\mu$	(93)	SPL	
(33)	$\Gamma{*}\omega \longrightarrow {*},\omega$	(94)	PTS	
(34)	$\Gamma{*}\square \longrightarrow {*}$	(95)	composition of two formulas of AT-1	
(35)	$\Gamma{*}, \longrightarrow {*}\delta,$	(96)	PTS	
(36)	$\Gamma{*}\mu, \longrightarrow {*}\delta\mu,$	(97)	PTS	
(37)	$\Gamma{*}\delta \longrightarrow {*}\theta$	(98)	PTS	
(38)	$\Gamma{*}\mu \longrightarrow \Gamma{*}\delta\mu$	(99)	PTS	
(39)	$\theta \longrightarrow\cdot$	(100)	AT-1	
(40)	$\longrightarrow \alpha$	(101)	SPL	

It was shown in STEP 3, that $\mathfrak{N}^{PP}$ transforms the ✱-pair □I,0✱II□I into the inverse partial product string $(P\uparrow\square)^{-1}$ ≐ 0,I00,II. By means of a normal algorithm $\mathfrak{N}^{PL}$, (whose scheme is given below) we shall obtain the ✱-pair whose left ✱-term is the corresponding palindrome string I,II,I left delimited by □ and the right ✱-term is the inverse partial product string 0,I00,II .

$\mathfrak{N}^{PL}$:

		Formula number
$\longrightarrow$ ✱		(0)
$\longrightarrow$ L	(L = I,II,I)	(1)
$\longrightarrow\cdot$ □		(2)

$\mathfrak{N}^{PL}$: 0,I00,II ⊢ ✱0,I00,II ⊢ I,II,I✱0,I00,II ⊢· □I,II,I✱0,I00,II

Now, $\mathfrak{N}^{CA}$ transforms □I,II,I✱0,I00,II into the string $Z\uparrow,$ ≐ II,I,0 where $Z\uparrow,$ corresponds to the convolved output sequence z(n) = 2,1,0. [NOTE: We have made use of the program LINCON in obtaining all the 306 elementary transformations which are given below]

$\mathfrak{N}^{CA}$: □I,II,I✱0,I00,II

LINCON :
ADJOIN PALINDROME STRING TO THE PARTIAL PRODUCT STRING, CHOOSE AND ADD:

$\square 1,11,1{*}0,100,11 \quad \leftarrow \quad [\ \mathcal{N}^{PL}:\ 0,100,11 \vdash \square 1,11,1{*}0,100,11\ ]$

```
101   α□1,11,1*0,100,11
101   αα□1,11,1*0,100,11
 67   β□1,11,1*0,100,11
 73   α□β1,11,1*0,100,11
 66   α□α1β,11,1*0,100,11
 66   α□α1α,β11,1*0,100,11
 66   α□α1α,α1β1,1*0,100,11
 66   α□α1α,α1α1β,1*0,100,11
 66   α□α1α,α1α1α,β1*0,100,11
 66   α□α1α,α1α1α,α1β*0,100,11
101   αα□α1α,α1α1α,α1β*0,100,11
 67   β□α1α,α1α1α,α1β*0,100,11
 73   α□βα1α,α1α1α,α1β*0,100,11
 72   α□β1α,α1α1α,α1β*0,100,11
 66   α□α1βα,α1α1α,α1β*0,100,11
 72   α□α1β,α1α1α,α1β*0,100,11
 66   α□α1α,βα1α1α,α1β*0,100,11
 72   α□α1α,β1α1α,α1β*0,100,11
 66   α□α1α,α1βα1α,α1β*0,100,11
 72   α□α1α,α1β1α,α1β*0,100,11
 66   α□α1α,α1α1βα,α1β*0,100,11
 72   α□α1α,α1α1β,α1β*0,100,11
 66   α□α1α,α1α1α,βα1β*0,100,11
 72   α□α1α,α1α1α,β1β*0,100,11
 66   α□α1α,α1α1α,α1ββ*0,100,11
 68   α□α1α,α1α1α,α1Γ*0,100,11
 61   α□α1α,α1α1Γ,α*0,100,11
 61   α□α1α,α1Γ1α,α*0,100,11
 61   α□α1α,α1Γ1α1α,α*0,100,11
 61   α□α1Γ,α1α1α,α*0,100,11
 61   α□α1Γ1α,α1α1α,α*0,100,11
 61   α1Γ□α1α,α1α1α,α*0,100,11
 65   1Γα□α1α,α1α1α,α*0,100,11
 69   1Γ□α1α,α1α1α,α*0,100,11
 71   1□Γα1α,α1α1α,α*0,100,11
 69   1□Γ1α,α1α1α,α*0,100,11
 70   1□1Γα,α1α1α,α*0,100,11
 69   1□1Γ,α1α1α,α*0,100,11
 70   1□1,Γα1α1α,α*0,100,11
 69   1□1,Γ1α1α,α*0,100,11
 70   1□1,1Γα1α,α*0,100,11
 69   1□1,1Γ1α,α*0,100,11
 70   1□1,11Γα,α*0,100,11
 69   1□1,11Γ,α*0,100,11
 70   1□1,11,Γα*0,100,11
 69   1□1,11,Γ*0,100,11
 97   1□1,11,*δ0,100,11
 80   1□1,11,*,δ0100,11
```

80 |□|,||,✱,|δ□□□,||
70 |□|,||,✱,|□δ□□,||
80 |□|,||,✱,|□□δ□,||
80 |□|,||,✱,|□□,δ□||
80 |□|,||,✱,|□□,|δ□|
80 |□|,||,✱,|□□,||δ□
86 |□|,||,β✱,,|□□,||δ□
87 |□|,||,β✱,|□□,||δ□
101 α|□|,||,β✱,|□□,||δ□
101 αα|□|,||,β✱,|□□,||δ□
67 β|□|,||,β✱,|□□,||δ□
66 α|β□|,||,β✱,|□□,||δ□
73 α|α□β|,||,β✱,|□□,||δ□
66 α|α□α|β,||,β✱,|□□,||δ□
66 α|α□α|α,β||,β✱,|□□,||δ
66 α|α□α|α,α|β|,β✱,|□□,||δ□
66 α|α□α|α,α|α|β,β✱,|□□,||δ□
66 α|α□α|α,α|α|α,ββ✱,|□□,||δ□
68 α|α□α|α,α|α|α,Γ✱,|□□,||δ□
61 α|α□α|α,α|α,Γ|α✱,|□□,||δ□
61 α|α□α|α,α,Γ|α|α✱,|□□,||δ□
61 α|α□α|α,Γ,α|α|α✱,|□□,||δ□
61 α|α□α,Γ|α,α|α|α✱,|□□,||δ□
61 α|α,Γ□α|α,α|α|α✱,|□□,||δ□
61 α,Γ|α□α|α,α|α|α✱,|□□,||δ□
65 ,Γα|α□α|α,α|α|α✱,|□□,||δ□
69 ,Γ|α□α|α,α|α|α✱,|□□,||δ□
70 ,|Γα□α|α,α|α|α✱,|□□,||δ□
69 ,|Γ□α|α,α|α|α✱,|□□,||δ□
71 ,|□Γα|α,α|α|α✱,|□□,||δ□
69 ,|□Γ|α,α|α|α✱,|□□,||δ□
70 ,|□|Γα,α|α|α✱,|□□,||δ□
69 ,|□|Γ,α|α|α✱,|□□,||δ□
70 ,|□|,Γα|α|α✱,|□□,||δ□
69 ,|□|,Γ|α|α✱,|□□,||δ□
70 ,|□|,|Γα|α✱,|□□,||δ□
69 ,|□|,|Γ|α✱,|□□,||δ□
70 ,|□|,||Γα✱,|□□,||δ□
69 ,|□|,||Γ✱,|□□,||δ□
96 ,|□|,||✱δ,|□□,||δ□
80 ,|□|,||✱|δ,□□,||δ□
79 ,|□|,||✱|□δ,□,||δ□
80 ,|□|,||✱|□□δ,,||δ□
80 ,|□|,||✱|□□,δ,||δ□
80 ,|□|,||✱|□□,|δ,|δ□
80 ,|□|,||✱|□□,||δ,δ□
101 α,|□|,||✱|□□,||δ,δ□
101 αα,|□|,||✱|□□,||δ,δ□
67 β,|□|,||✱|□□,||δ,δ□
66 α,β|□|,||✱|□□,||δ,δ□
66 α,α|β□|,||✱|□□,||δ,δ□
73 α,α|α□β|,||✱|□□,||δ,δ□
66 α,α|α□α|β,||✱|□□,||δ,δ□
66 α,α|α□α|α,β||✱|□□,||δ,δ□

66 α,α |α□α |α,α |β |✱ |□□,||δ,δ□
66 α,α |α□α |α,α |α |β✱ |□□,||δ,δ□
101 αα,α |α□α |α,α |α |β✱ |□□,||δ,δ□
67 β,α |α□α |α,α |α |β✱ |□□,||δ,δ□
66 α,βα |α□α |α,α |α |β✱ |□□,||δ,δ□
72 α,β |α□α |α,α |α |β✱ |□□,||δ,δ□
66 α,α |βα□α |α,α |α |β✱ |□□,||δ,δ□
72 α,α |β□α |α,α |α |β✱ |□□,||δ,δ□
73 α,α |α□βα |α,α |α |β✱ |□□,||δ,δ□
72 α,α |α□β |α,α |α |β✱ |□□,||δ,δ□
66 α,α |α□α |βα,α |α |β✱ |□□,||δ,δ□
72 α,α |α□α |β,α |α |β✱ |□□,||δ,δ□
66 α,α |α□α |α,βα |α |β✱ |□□,||δ,δ□
72 α,α |α□α |α,β |α |β✱ |□□,||δ,δ□
66 α,α |α□α |α,α |βα |β✱ |□□,||δ,δ□
72 α,α |α□α |α,α |β |β✱ |□□,||δ,δ□
66 α,α |α□α |α,α |α |ββ✱ |□□,||δ,δ□
68 α,α |α□α |α,α |α |Γ✱ |□□,||δ,δ□
61 α,α |α□α |α,α |Γ |α✱ |□□,||δ,δ□
61 α,α |α□α |α |Γ,α |α✱ |□□,||δ,δ□
61 α,α |α□α |Γ |α,α |α✱ |□□,||δ,δ□
61 α,α |α |Γ□α |α,α |α✱ |□□,||δ,δ□
61 α,α |Γ |α□α |α,α |α✱ |□□,||δ,δ□
61 α |Γ,α |α□α |α,α |α✱ |□□,||δ,δ□
65 |Γα,α |α□α |α,α |α✱ |□□,||δ,δ□
69 |Γ,α |α□α |α,α |α✱ |□□,||δ,δ□
70 |,Γα |α□α |α,α |α✱ |□□,||δ,δ□
69 |,Γα□α |α,α |α✱ |□□,||δ,δ□
70 |,|Γα□α |α,α |α✱ |□□,||δ,δ□
69 |,|Γ□α |α,α |α✱ |□□,||δ,δ□
71 |,|□Γα |α,α |α✱ |□□,||δ,δ□
69 |,|□Γ |α,α |α✱ |□□,||δ,δ□
70 |,|□|Γα,α |α✱ |□□,||δ,δ□
69 |,|□|Γ,α |α✱ |□□,||δ,δ□
70 |,|□|,Γα |α✱ |□□,||δ,δ□
69 |,|□|,Γ |α✱ |□□,||δ,δ□
70 |,|□|,|Γα✱ |□□,||δ,δ□
69 |,|□|,|Γ✱ |□□,||δ,δ□
99 |,|□|,|Γ✱δ |□□,||δ,δ□
79 |,|□|,|Γ✱□δ |□,||δ,δ□
80 |,|□|,|Γ✱□□δ |,||δ,δ□
80 |,|□|,|Γ✱□□,δ |||δ,δ□
80 |,|□|,|Γ✱□□,|δ ||δ,δ□
80 |,|□|,|Γ✱□□,||δ |δ,δ□
95 |,|□|,|✱□,||δ |δ,δ□
101 α |,|□|,|✱□,||δ |δ,δ□
101 αα |,|□|,|✱□,||δ |δ,δ□
67 β |,|□|,|✱□,||δ |δ,δ□
66 α |β,|□|,|✱□,||δ |δ,δ□
66 α |α,β |□|,|✱□,||δ |δ,δ□
66 α |α,α |β□|,|✱□,||δ |δ,δ□
73 α |α,α |α□β |,|✱□,||δ |δ,δ□
66 α |α,α |α□α |β,|✱□,||δ |δ,δ□
66 α |α,α |α□α |α,β |✱□,||δ |δ,δ□

66	α|α,α|α□α|α,α|β✱□,||δ|δ,δ□
101	αα|α,α|α□α|α,α|β✱□,||δ|δ,δ□
67	β|α,α|α□α|α,α|β✱□,||δ|δ,δ□
66	α|βα,α|α□α|α,α|β✱□,||δ|δ,δ□
72	α|β,α|α□α|α,α|β✱□,||δ|δ,δ□
66	α|α,βα|α□α|α,α|β✱□,||δ|δ,δ□
72	α|α,β|α□α|α,α|β✱□,||δ|δ,δ□
66	α|α,α|βα□α|α,α|β✱□,||δ|δ,δ□
72	α|α,α|β□α|α,α|β✱□,||δ|δ,δ□
73	α|α,α|α□βα|α,α|β✱□,||δ|δ,δ□
72	α|α,α|α□β|α,α|β✱□,||δ|δ,δ□
66	α|α,α|α□α|βα,α|β✱□,||δ|δ,δ□
72	α|α,α|α□α|β,α|β✱□,||δ|δ,δ□
66	α|α,α|α□α|α,βα|β✱□,||δ|δ,δ□
72	α|α,α|α□α|α,β|β✱□,||δ|δ,δ□
66	α|α,α|α□α|α,α|ββ✱□,||δ|δ,δ□
68	α|α,α|α□α|α,α|Γ✱□,||δ|δ,δ□
61	α|α,α|α□α|α|Γ,α✱□,||δ|δ,δ□
61	α|α,α|α□α|Γ|α,α✱□,||δ|δ,δ□
61	α|α,α|α|Γ□α|α,α✱□,||δ|δ,δ□
61	α|α,α|Γ|α□α|α,α✱□,||δ|δ,δ□
61	α|α|Γ,α|α□α|α,α✱□,||δ|δ,δ□
61	α|Γ|α,α|α□α|α,α✱□,||δ|δ,δ□
65	|Γα|α,α|α□α|α,α✱□,||δ|δ,δ□
69	|Γ|α,α|α□α|α,α✱□,||δ|δ,δ□
70	||Γα,α|α□α|α,α✱□,||δ|δ,δ□
69	||Γ,α|α□α|α,α✱□,||δ|δ,δ□
70	||,Γα|α□α|α,α✱□,||δ|δ,δ□
69	||,Γ|α□α|α,α✱□,||δ|δ,δ□
70	||,|Γα□α|α,α✱□,||δ|δ,δ□
69	||,|Γ✱pα|α,α✱□,||δ|δ,δ□
71	||,|□Γα|α,α✱□,||δ|δ,δ□
69	||,|□Γ|α,α✱□,||δ|δ,δ□
70	||,|□|Γα,α✱□,||δ|δ,δ□
69	||,|□|Γ,α✱□,||δ|δ,δ□
70	||,|□|,Γα✱□,||δ|δ,δ□
69	||,|□|,Γ✱□,||δ|δ,δ□
97	||,|□|,✱δ□,||δ|δ,δ□
80	||,|□|,✱,δ□||δ|δ,δ□
80	||,|✱p|,✱,|δ□|δ|δ,δ□
80	||,|□|,✱,||δ□δ|δ,δ□
86	||,|□|,β✱,,||δ□δ|δ,δ□
87	||,|□|,β✱,||δ□δ|δ,δ□
101	α||,|□|,β✱,||δ□δ|δ,δ□
101	αα||,|□|,β✱,||δ□δ|δ,δ□
67	β||,|□|,β✱,||δ□δ|δ,δ□
66	α|β|,|□|,β✱,||δ□δ|δ,δ□
66	α|α|β,|□|,β✱,||δ□δ|δ,δ□
66	α|α|α,β|□|,β✱,||δ□δ|δ,δ□
66	α|α|α,α|β□|,β✱,||δ□δ|δ,δ□
73	α|α|α,α|α□β|,β✱,||δ□δ|δ,δ□
66	α|α|α,α|α□α|β,β✱,||δ□δ|δ,δ□
66	α|α|α,α|α□α|α,ββ✱,||δ□δ|δ,δ□
68	α|α|α,α|α□α|α,Γ✱,||δ□δ|δ,δ□

61 α|α|α,α|α□α,Γ|α✱,||δ□δ|δ,δ□
61 α|α|α,α|α,Γ□α|α✱,||δ□δ|δ,δ□
61 α|α|α,α,Γ|α□α|α✱,||δ□δ|δ,δ□
61 α|α|α,Γ,α|α□α|α✱,||δ□δ|δ,δ□
61 α|α,Γ|α,α|α□α|α✱,||δ□δ|δ,δ□
61 α,Γ|α|α,α|α□α|α✱,||δ□δ|δ,δ□
65 ,Γα|α|α,α|α□α|α✱,||δ□δ|δ,δ□
69 ,Γ|α|α,α|α□α|α✱,||δ□δ|δ,δ□
70 ,|Γα|α,α|α□α|α✱,||δ□δ|δ,δ□
69 ,|Γ|α,α|α□α|α✱,||δ□δ|δ,δ□
70 ,||Γα,α|α□α|α✱,||δ□δ|δ,δ□
69 ,||Γ,α|α□α|α✱,||δ□δ|δ,δ□
70 ,||,Γα|α□α|α✱,||δ□δ|δ,δ□
69 ,||,Γ|α□α|α✱,||δ□δ|δ,δ□
70 ,||,|Γα□α|α✱,||δ□δ|δ,δ□
69 ,||,|Γ□α|α✱,||δ□δ|δ,δ□
71 ,||,|□Γα|α✱,||δ□δ|δ,δ□
69 ,||,|□Γ|α✱,||δ□δ|δ,δ□
70 ,||,|□|Γα✱,||δ□δ|δ,δ□
69 ,||,|□|Γ✱,||δ□δ|δ,δ□
96 ,||,|□|✱δ,||δ□δ|δ,δ□
80 ,||,|□|✱|δ,|δ□δ|δ,δ□
80 ,||,|□|✱||δ,δ□δ|δ,δ□
101 α,||,|□|✱||δ,δ□δ|δ,δ□
101 αα,||,|□|✱||δ,δ□δ|δ,δ□
67 β,||,|□|✱||δ,δ□δ|δ,δ□
66 α,β||,|□|✱||δ,δ□δ|δ,δ□
66 α,α|β|,|□|✱||δ,δ□δ|δ,δ□
66 α,α|α|β,|□|✱||δ,δ□δ|δ,δ□
66 α,α|α|α,β|□|✱||δ,δ□δ|δ,δ□
66 α,α|α|α,α|β□|✱||δ,δ□δ|δ,δ□
73 α,α|α|α,α|α□β|✱||δ,δ□δ|δ,δ□
66 α,α|α|α,α|α□α|β✱||δ,δ□δ|δ,δ□
101 αα,α|α|α,α|α□α|β✱||δ,δ□δ|δ,δ□
67 β,α|α|α,α|α□α|β✱||δ,δ□δ|δ,δ□
66 α,βα|α|α,α|α□α|β✱||δ,δ□δ|δ,δ□
72 α,β|α|α,α|α□α|β✱||δ,δ□δ|δ,δ□
66 α,α|βα|α,α|α□α|β✱||δ,δ□δ|δ,δ□
72 α,α|β|α,α|α□α|β✱||δ,δ□δ|δ,δ□
66 α,α|α|βα,α|α□α|β✱||δ,δ□δ|δ,δ□
72 α,α|α|β,α|α□α|β✱||δ,δ□δ|δ,δ□
66 α,α|α|α,βα|α□α|β✱||δ,δ□δ|δ,δ□
72 α,α|α|α,β|α□α|β✱||δ,δ□δ|δ,δ□
66 α,α|α|α,α|βα□α|β✱||δ,δ□δ|δ,δ□
72 α,α|α|α,α|β□α|β✱||δ,δ□δ|δ,δ□
73 α,α|α|α,α|α□βα|β✱||δ,δ□δ|δ,δ□
72 α,α|α|α,α|α□β|β✱||δ,δ□δ|δ,δ□
66 α,α|α|α,α|α□α|ββ✱||δ,δ□δ|δ,δ□
68 α,α|α|α,α|α□α|Γ✱||δ,δ□δ|δ,δ□
61 α,α|α|α,α|α|Γ□α✱||δ,δ□δ|δ,δ□
61 α,α|α|α,α|Γ|α□α✱||δ,δ□δ|δ,δ□
61 α,α|α|α|Γ,α|α□α✱||δ,δ□δ|δ,δ□
61 α,α|α|Γ|α,α|α□α✱||δ,δ□δ|δ,δ□
61 α,α|Γ|α|α,α|α□α✱||δ,δ□δ|δ,δ□

```
 61   α|Γ,α|α|α,α|α□α✱||δ,δ□δ|δ,δ□
 65   |Γα,α|α|α,α|α□α✱||δ,δ□δ|δ,δ□
 69   |Γ,α|α|α,α|α□α✱||δ,δ□δ|δ,δ□
 70   |,Γα|α|α,α|α□α✱||δ,δ□δ|δ,δ□
 69   |,Γ|α|α,α|α□α✱||δ,δ□δ|δ,δ□
 70   |,|Γα|α,α|α□α✱||δ,δ□δ|δ,δ□
 69   |,|Γ|α,α|α□α✱||δ,δ□δ|δ,δ□
 70   |,||Γα,α|α□α✱||δ,δ□δ|δ,δ□
 69   |,||Γ,α|α□α✱||δ,δ□δ|δ,δ□
 70   |,||,Γα|α□α✱||δ,δ□δ|δ,δ□
 69   |,||,Γ|α□α✱||δ,δ□δ|δ,δ□
 70   |,||,|Γα□α✱||δ,δ□δ|δ,δ□
 69   |,||,|Γ□α✱||δ,δ□δ|δ,δ□
 71   |,||,|□Γα✱||δ,δ□δ|δ,δ□
 69   |,||,|□[illegible]✱||δ,δ□δ|δ,δ□
 99   |,||,|□[illegible]✱δ||δ,δ□δ|δ,δ□
 80   |,||,|□Γ✱|δ|δ,δ□δ|δ,δ□
 99   |,||,|□Γ✱δ|δ|δ,δ□δ|δ,δ□
 98   |,||,|□✱θ|δ|δ,δ□δ|δ,δ□
 75   |,||,|✱θ|δ|δ,δ□δ|δ,δ□
 74   |,||,✱θ|δ|δ,δ□δ|δ,δ□
 74   |,||✱θ|δ|δ,δ□δ|δ,δ□
 74   |,|✱θ|δ|δ,δ□δ|δ,δ□
 74   |,✱θ|δ|δ,δ□δ|δ,δ□
 74   |✱θ|δ|δ,δ□δ|δ,δ□
 74   ✱θ|δ|δ,δ□δ|δ,δ□
 76   θ|δ|δ,δ□δ|δ,δ□
 78   |θδ|δ,δ□δ|δ,δ□
 77   |θ|δ,δ□δ|δ,δ□
 78   ||θδ,δ□δ|δ,δ□
 77   ||θ,δ□δ|δ,δ□
 78   ||,θδ□δ|δ,δ□
 77   ||,θ□δ|δ,δ□
 78   ||,□θδ|δ,δ□
 77   ||,□θ|δ,δ□
 78   ||,□|θδ,δ□
 77   ||,□|θ,δ□
 78   ||,□|,θδ□
 77   ||,□|,θ□
 78   ||,□|,□θ
 90   ||,|,□θ
100   ||,|,□
```

The string $Zt_1 \doteq$ ||,|,□ is obtained after 306 elementary transformations .

STEP 5

One can construct another normal algorithm $\mathcal{N}^{DC}$ over the alphabet $\mathcal{A}_0 \cup \mathfrak{D}_1 \cup \mathbb{N}$ that decodes Zt_1 into the convolved output z(n). $\mathcal{N}^{DC}$ (||,|,□) $\doteq$ 2,1,0

So far, we outlined the construction details of certain normal algorithms

which form the system $\mathfrak{R}^{con}$. In what follows, we indicate the type of maps corresponding to those normal algorithms and show how they are combined to form the system $\mathfrak{R}^{con}$. [Table 4.3.3 and Fig.4.3.1]

For convenience, we recall from STEP 1, the alphabets over the union of which $\mathfrak{R}^{con}$ has been constructed :

$\mathcal{A}_0 = \{0\ 1\}$; $\mathfrak{D}_1 = \{,\}$; $\mathfrak{D}_2 = \{\square\}$; $\mathfrak{D}_3 = \{*\}$ and $\mathcal{E} = \{ a\ b\ \alpha\ \beta\ \Gamma\ \delta\ \theta\ \sigma\ \omega\ \nu \}$.

$\mathfrak{D} = \mathfrak{D}_1 \cup \mathfrak{D}_2 \cup \mathfrak{D}_3$; $\mathcal{A}_{11} = \mathcal{A}_0 \cup \mathfrak{D}_1$; $\mathcal{A}_{12} = \mathcal{A}_0 \cup \mathfrak{D}_2$; $\mathcal{A}_{13} = \mathcal{A}_{11} \cup \mathcal{A}_{12}$;

$\mathcal{A}_{14} = \mathcal{A}_{13} \cup \mathfrak{D}_3$; $A_0 = \{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\}$.

Let us also consider N = {x | x is a natural number in the traditional sense.} and $\mathbb{S}$ = {x | x is a non-negative integer sequence}.

Table 4.3.3: Types of maps corresponding to the normal algorithms that constitute $\mathfrak{R}^{con}$

Sl. No.	Normal algorithm	Alphabet over which constructed	Type of the map	Function
1	$\mathcal{N}^{\dagger,}$	$A_0 \cup \mathcal{A}_{11}$	$\mathbb{S} \longrightarrow X_{11}$	forms ,-diluted strings from $\mathcal{A}_0$ representing natural number sequences.
2	$\mathcal{N}^{\dagger\square}$	$A_0 \cup \mathcal{A}_{12}$	$\mathbb{S} \longrightarrow X_{12}$	forms $\square$-diluted strings from $\mathcal{A}_0$ representing natural number sequences.
3	$\mathcal{N}^{ID}$	any alphabet $\mathcal{A}$	$\mathcal{A}^* \longrightarrow \mathcal{A}^*$	transforms any string to the same string.
4	$\mathcal{N}^{*}$	$\mathcal{A}_{14}$	$X'_{13} \longrightarrow X_{14}$	forms the pre-convolution string.
5	$\mathcal{N}^{PP}$	$\mathcal{A}_{14}$	$X_{14} \longrightarrow X''_{13}$	transforms pre-convolution

contd.

				string into inverse partial product string.
6	$\mathcal{N}^{PL}$	$\mathcal{A}_{14}$	$X''_{13} \longrightarrow X_{14}$	forms the ✱-pair whose left ✱-term is the suitable palindrome string with □ as the left delimiter and whose right ✱-term is the inverse partial product string.
7	$\mathcal{N}^{CA}$	$\mathcal{A}_{14}$	$X_{14} \longrightarrow X_{11}$	transforms the ✱-pair formed by $\mathcal{N}^{PL}$ into a string corresponding to the convolved output sequence.
8	$\mathcal{N}^{DC}$	$A_0 \cup \mathcal{A}_{11}$	$X_{11} \longrightarrow S$	decodes the string due to $\mathcal{N}^{CA}$ into the corresponding non-negative integer sequence.

The block diagram of the operational scheme of the constructive system $\Re^{con}$ is shown in figure 4.3.1.

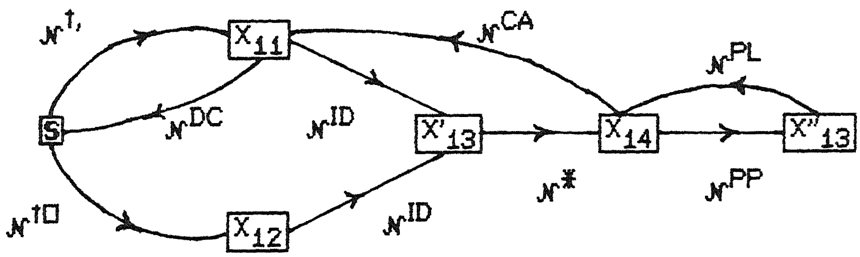


FIGURE 4.3.1: Operational scheme of $\Re^{con}$

PART - II

SECTION 5

STUDY OF NORMAL ALGORITHMIC SIGNAL PROCESSING SYSTEMS IN TERMS OF FORMAL LANGUAGES GENERATED BY A GRAMMAR

The purpose of this section is to introduce a formal language for the study of constructive (normal algorithmic) signal processing systems. The motivation for such a study is provided by the fact that a constructive signal processing system $\mathfrak{R}$ over an alphabet $\mathcal{A}$, is in essence a mapping that recognizes a subset X (i.e., a language) of the free monoid $\mathcal{A}^*$ and maps it onto another subset Y of words in $\mathcal{A}^*$. With X and Y as the input and output signal spaces, $\mathfrak{R}$ admits of being treated as a system described by a grammar and also as an automaton.

5.1 FORMAL REPRESENTATION OF SIGNALS AND SYSTEMS OVER ALPHABETS

The starting point of our study is the *formal representability* of words and algorithms over alphabets by *Elementary Formal Systems* (EFS) of Smullyan [33].

An *elementary formal system* over an alphabet $\mathcal{A}$ is defined as a set of alphabets, $\mathcal{A}$, $\mathcal{V}$, $\mathcal{P}$ and $\mathcal{I}$, together with a set of axioms. $\mathcal{A}$ is a given alphabet whose words are the basic elements of interest in the EFS. $\mathcal{V}$ is an alphabet of two types of variables, the first type consisting of *proper variables* that range over the free semigroup $\mathcal{A}^+$, and the other type consisting of *improper variables* that range over the free monoid $\mathcal{A}^*$. $\mathcal{P}$ is an alphabet of *predicates*, each of which assigned with a unique positive integer called its *degree*. $\mathcal{I}$ is an alphabet containing two symbols called the *implication sign* and *punctuation sign*. The set of axioms of the system are strings from the above distinct alphabets and are known as *well formed formulas* (wff).

Any string of symbols from the alphabet $\mathcal{A} \cup \mathcal{V}$ is known as a *term* generally

denoted by t, of the system. By an *atomic formula* F, we mean a string from the alphabet $\mathcal{A} \cup \mathcal{V} \cup \mathcal{P}$ of the form $pt_1t_2 \ldots\ldots t_m$ where p is a predicate of degree m and $t_1, t_2, \ldots\ldots, t_m$ are terms. A well formed formula is defined as a word $F\uparrow\rightarrow$ in $\mathcal{A} \cup \mathcal{V} \cup \mathcal{P} \cup \mathcal{I}$ where $F\uparrow\rightarrow = F_1 \rightarrow F_2 \rightarrow \ldots\ldots \rightarrow F_i \rightarrow \ldots\ldots \rightarrow F_n$ and $F_i : (1 \leq i \leq n)$ is an atomic formula. $F_1, F_2, \ldots\ldots, F_i \ldots\ldots, F_{n-1}$ are called as *premises* and F_n as the *conclusion*. Any atomic formula is also a wff which means that it is by itself a premiss and the conclusion. If all the variables of a wff are substituted by strings from $\mathcal{A}$, then the resulting string is known as an *instance* of the wff. A variable-free wff is known as a *sentence*. A *theorem* of an EFS is either an axiom or a string which is derivable from the set of axioms of the EFS with the help of the following *rules of inference* :

(i) Instantiation : Substitution of strings from $\mathcal{A}$ for variables.

(ii) Modus Ponens : The rule by which a formula can be inferred from another atomic formula.

Let p be a predicate of degree n from $\mathcal{P}$ and let W be a set of n-tuples of words from $\mathcal{A}^*$. Then, we say that p represents W if $px_1x_2 \ldots x_n$ is provable in the EFS over $\mathcal{A}$ for every n-tuple $x_1x_2 \ldots x_n$ in W.

As an illustration of an EFS, let us consider the alphabet $\mathcal{A}_0 = \{ 0 \mid \}$ in which the natural number system can be formally represented as shown below. The relevant alphabets for the EFS in this case are:

(i) $\mathcal{A}_0 = \{ 0 \mid \}$

(ii) $\mathcal{V} = \{ x \}$

(iii) $\mathcal{P} = \{ \mathbb{N} \}$, where $\mathbb{N}$ is of degree 1.

(iv) $\mathcal{I} = \{ \rightarrow \}$

and the axioms are:

(i) $\mathbb{N}0$ (i.e., in set-theoretic terms, $0 \in \mathbb{N}$)

(ii) $\mathbb{N}x \rightarrow \mathbb{N}x\mid$ (if the word represented by the variable

x is in $\mathbb{N}$ then the word xl is also in $\mathbb{N}$.)

Using these axioms, the system of natural numbers is formally represented over $\mathcal{A}_0$ as $\mathbb{N}$ = { 0, 0l, 0ll, 0lll, 0llll, ... }. $\mathbb{N}$ is called the system of *Constructive Natural Numbers* [20], [31].

REMARK: We note that in the above example we have used the same symbol $\mathbb{N}$ for denoting the predicate and the set of natural numbers.

Now, we show with the help of the following two propositions, that signal spaces and constructive signal processing systems are formally representable over finite alphabets. Here, we refer to subsets of a free monoid as signal spaces and algorithms such as associative calclui and normal alorithms as constructive systems.

PROPOSITION 5.1.1

Let $S_{\mathfrak{K}}$ be a subset of a free monoid $\mathcal{A}^*$ and let $\mathfrak{K}$ be an associative calculus that recognizes $S_{\mathfrak{K}}$. Then, $S_{\mathfrak{K}}$ is representable in an elementary formal system $E_{\mathfrak{K}}$ over $\mathcal{A}$. [Subsection 1.4]

PROOF:

Recall from section 1 that elements of the set $S_{\mathfrak{K}}$ are equivalents with reference to the defining system $\mathfrak{D}$ of $\mathfrak{K}$, where $\mathfrak{D}$ is an unordered list of $\mathfrak{I}$-class equivalence relations. We now, construct an EFS $E_{\mathfrak{K}}$ for representing the set $S_{\mathfrak{K}}$.

Alphabets :	(i)	$\mathcal{A}$	(a given alphabet)
	(ii)	$\mathcal{V}$ = { α β x y v w }	(α and β are improper variables and x, y, v and w are proper variables.)
	(iii)	$\mathcal{P}$ = { f r $\mathfrak{D}$ $S_{\mathfrak{K}}$}	($S_{\mathfrak{K}}$ is a predicate of degree 1 and f, r and $\mathfrak{D}$ are predicates of degree 2.)
	(iv)	$\mathcal{I}$ = { $\rightarrow$, }	(punctuation symbols)
Axioms :	(i)	$S_{\mathfrak{K}}x$:	(x is an element of $S_{\mathfrak{K}}$)
	(ii)	$fx,\alpha x\beta \rightarrow ryx \rightarrow \mathfrak{D}vw$	(if x is the factor of a word $\alpha x\beta$ then y is the replacement of x which in turn implies that a

word $v = \alpha x \beta$ is a $\mathfrak{D}$-equivalent of $w = \alpha y \beta$)

(iii) $fy,\alpha y\beta \rightarrow rxy \rightarrow \mathfrak{D}vw$ (if y is the factor of a word $\alpha y \beta$ then x is the replacement of y which in turn implies that a a word $v = \alpha y \beta$ is a $\mathfrak{D}$-equivalent of $w = \alpha x \beta$)

(iv) $\mathfrak{D}vw$ (v is a $\mathfrak{D}$-equivalent of w)

Let $\langle x_i , y_i \rangle$, $1 \leq i \leq n$ be the n ordered $\mathfrak{I}$-class relations of the defining system $\mathfrak{D}$ of an associative calculus $\mathfrak{R}$ For any pair of equivalent words v and w in $S_{\mathfrak{R}}$, $\mathfrak{D}vw$ is provable in $E_{\mathfrak{R}}$ by the inductive instantiation of axioms (ii), (iii) and (iv), using the $\mathfrak{I}$-class relations of $\mathfrak{D}$ Now, given any two arbitrary words v and w in $\mathcal{A}^+$, if $\mathfrak{D}vw$ is provable in $E_{\mathfrak{R}}$, then $v, w \in S_{\mathfrak{R}}$ Hence, $S_{\mathfrak{R}}$ is representable in an elementary formal system $E_{\mathfrak{R}}$ over $\mathcal{A}$.

PROPOSITION 5 1 2

Let $W_{\mathcal{N}}$ be a subset of a free monoid $\mathcal{A}^*$ and let $\mathcal{N}$ be a normal algorithm that recognizes $W_{\mathcal{N}}$ Then, $W_{\mathcal{N}}$ is representable in an EFS, $E_{\mathcal{N}}$ over $\mathcal{A}$

PROOF

As we recall from subsection 2 2, a normal algorithm $\mathcal{N}$ over an alphabet $\mathcal{A}$ is interpreted as an $\mathfrak{I}$-class presentation of the free monoid $\mathcal{A}^*$ defined by an ordered list of n $\mathfrak{I}$-class partial order relations corresponding to the n substitution formulas of $\mathcal{N}$ We now, construct an EFS, $E_{\mathcal{N}}$ for representing the set $W_{\mathcal{N}}$

Alphabets (i) $\mathcal{A}$ (the given alphabet)

(ii) $\mathcal{V} = \{ \alpha \ \beta \ x \ y \ v \ w \}$ (α and β are improper variables and x, y, v and w are proper variables)

(iii) $\mathcal{P} = \{ a \ f \ r \ W_{\mathcal{N}} \}$ (a and $W_{\mathcal{N}}$ are predicates of degree 1 and r and f are predicates of degree 2)

(iv) $\mathcal{I} = \{ \rightarrow , \}$ (punctuation symbols)

Axioms (i) $W_{\mathcal{N}}x$ (i e , set theoretically $x \in W_{\mathcal{N}}$)

(ii) $fx,\alpha x\beta \to ryx \to av$ (if x is a factor of a word $v = \alpha x \beta$ then y is a replacement of x in v which implies that v is amenable to $\mathcal{N}$)

(v) av (v is amenable to $\mathcal{N}$)

Let $\langle x_i , y_i \rangle$, $1 \le i \le n$ be the n ordered $\mathfrak{J}$-class partial order relations of the scheme $\mathfrak{S}$ of a normal algorithm $\mathcal{N}$ For a word v in $W_{\mathcal{N}}$, the axiom av is provable in $E_{\mathcal{N}}$ by the inductive instantiation of the axioms (ii) and (iii) using the relations of $\mathfrak{S}$ Now, given a word v in $\mathcal{A}^*$, if av is provable in $E_{\mathcal{N}}$ then $v \in W_{\mathcal{N}}$ Hence, $W_{\mathcal{N}}$ is formally representable in $E_{\mathcal{N}}$ This completes the proof

Since $S_{\mathfrak{R}}$ and $W_{\mathcal{N}}$ are formally representable in the elementary formal systems $E_{\mathfrak{R}}$ and $E_{\mathcal{N}}$ respectively, the classes of associative calculi and normal algorithms are formally representable over an alphabet $\mathcal{A}$

5 2 FORMULATION OF A STRING MANIPULATION LANGUAGE FOR CONSTRUCTIVE SIGNAL PROCESSING

Since subsets of a free monoid and algorithms such as associative calculi and normal algorithms are formally representable in EFS' over alphabets, the possibility exists of studying signal spaces and constructive systems in an EFS type language We find that Fitting's notion of EFS type string manipulation languages [14], is useful for this purpose Before attempting to formulate one such language for constructive signal processing, we review some of the basic details regarding the EFS language of Fitting

5 2 1 FITTING'S BASIC STRING MANIPULATION LANGUAGE EFS(str(L))

EFS(str(L)) is a representative example of an EFS language for a data structure of character strings over an alphabet, say L

DEFINITION 5 2 1 1 [14]

EFS(str(L)) over an alphabet L is defined to be a collection of the following:

(i) an alphabet L of distinct symbols

(ii) a data structure $< L^*, CON_L >$ where L^* is the free monoid of L and CON_L is a 3-place relation $\langle u,v,w \rangle$ such that for any three $u,v,w \in L^*$, w is the word obtained by right concatenating v to u

(iii) a set of identifiers which are words consisting of capital letters from the English alphabet

(iv) an alphabet consisting of variables denoted by x, y, z etc , and

(v) an alphabet consisting of punctuation symbols · { → ( ) , }

The notions *term*, *atomic statement* and *wff* of EFS(str(L)) have to be understood as they have been defined already in subsection 5 1

A work space W of an EFS , in general, consists of the following

(i) a specification of a *domain*

(ii) a list of *reserved identifiers* (The term Identifiers refers to predicate labels A reserved identifier will not occur in the position of the conclusion of a statement (wff))

(iii) a specification of what relations the reserved identifiers represent Such relations are the *given relations* of W.

Every elementary formal system is associated with its *basic work space* For instance, the basic work space of EFS(str(L)) consists of the following

(i) the domain L^*

(ii) the only reserved identifier CON .

(iii) the label CON represents the concatenation relation CON_L on L

A *procedure* P in a work space W is a list of acceptable statements written one below the other like a PASCAL program The first statement is known as the *header* and the block of remaining statements as the *body* of the procedure The header designates an unreserved identifier together with degree *n* of the predicate for which the identifier is the label The identifier in the header

represents the name of the procedure and its output as well

EXAMPLE 5 2 1 1

EFS language EFS(str(L))

Data structure $< L^*, CON_L >$

Work space (W)

(i) Elements of L^*

(ii) CON (the reserved identifier with degree 3)

(iii) CON represents concatenation of an element v to the right of another u such that $uv=w$, $u,v,w \in L^*$

Name of the procedure FAC(2)

FAC(2)

FAC(x,x);

FAC(x,Λ),

$CON(x,y,z) \to CON(v,z,y) \to FAC(y,x)$

The language recognized by the procedure FAC is $L^* x L^*$

The computation of a procedure P in a work space W gives rise to sequences of statements of the following kinds

(i) a substitution instance of a procedure statement

(ii) an instance of a given relation of W

(iii) an assignment statement T where its previous line is

$X_1 \to X_2 \to \ldots \to T$ and $X_1, X_2, \ldots, T$ are atomic statements.

A sequence of statements resulting from a computation of a procedure P is known as a *trace*, which is analogous to what is called a *derivation* in mathematical logic A *restricted trace* is the sequence of statements governed by the first two conditions for a trace and by a third condition known as the *restricted assignment rule* This rule allows statements of the form $X \to F$ where X is atomic and the

assignment F need not be

The basic work space of an EFS can be expanded by adding to it more and more reserved identifiers as needed For example, the basic work space W of EFS(str(L)) could be expanded as W' by adding FAC as a reserved identifier Now we shall construct a procedure GREQ in W' which would compute the graphical equivalence between two words, in the following manner

Let the basic alphabet be L = { 0 1 }

GREQ(2).

GREQ(x,x),

FAC(y,x)→FAC(x,y)→GREQ(x,y)

A trace of GREQ would be of the form

GREQ(2)

GREQ(101,101);

FAC(101,101)→FAC(101,101)→GREQ(101,101)

EFS(str(L)) has an enormous language (subset) recognizing power This can be inferred from the following argument

Let **P** be the class of all possible procedures which could be constructed in W of EFS(str(L)) Then, **P** is said to consist of *basic* or *first level* procedures The procedures which are constructible only in W' and not in W are called *second level* procedures As already mentioned, W' is an expanded work space of W with an additional reserved identifier which has been used previously as an unreserved identifier in W Every such unreserved identifier, let us call it " i ", when individually added to W as a reserved identifier, forms an expanded work space W'_i in which a class $\mathbf{P}'_i$ of second level procedures can be constructed Likewise, the basic work space W along with two distinct reserved identifiers " i " and " j ", forms the work space W'_{ij} in which $\mathbf{P}'_{ij}$, a class of *third level* procedures can be constructed. The generalization of the basic work space expansion in the above

manner exhibits the language recognizing power of EFS(str(L))

5.2.2 THE STRING MANIPULATION LANGUAGE EFS(spl($\mathcal{A}$)) FOR CONSTRUCTIVE SIGNAL PROCESSING

With the material covered in 5.2.1 as the required background, we shall now formulate a string manipulation language EFS(spl($\mathcal{A}$)) for constructive signal processing

DEFINITION 5.2.2 :

EFS(spl($\mathcal{A}$)) over an alphabet $\mathcal{A}$ is defined to be the collection of the following

(i) an alphabet $\mathcal{A}$ of n distinct symbols

(ii) a data structure $< \mathcal{A}^*, CON_{\mathcal{A}}, SBT_{\mathcal{A}}>$ where $\mathcal{A}^*$ is the free monoid of $\mathcal{A}$, $CON_{\mathcal{A}}$ is a 3-place relation $\langle u,v,w\rangle$ such that w is the concatenated string uv : $(u,v \in \mathcal{A}^*)$, and $SBT_{\mathcal{A}}$ is a 2-place relation $\langle u,v\rangle$ such that v is substitute of u where $u,v \in \mathcal{A}^*$

(iii) a set of identifiers denoting predicate labels [A break symbol " _ " could be used in writing identifier labels For example, FAC_OF is a valid identifier whereas " FAC_ " or " _OF " are not]

(iv) an alphabet consisting of variables of two types (1) u,v,w,x,y,z which range over $\mathcal{A}^*$ and (2) ξ,μ,η which range over $\mathcal{A}$ [The variables could be subscripted or superscripted as per requirement]

(v) an alphabet consisting of punctuation symbols { $\rightarrow$ ( ) , }

The basic work space W of EFS(spl($\mathcal{A}$)) consists of the following

(i) the domain $\mathcal{A}^*$

(ii) CON and SBT are the only reserved identifiers

(iii) CON represents the concatenation relation $CON_{\mathcal{A}}$ on $\mathcal{A}$;

SBT represents the substitution relation $SBT_{\mathcal{A}}$ on $\mathcal{A}$

In general, the output of an EFS procedure is called a *generated relation* Let R be a reserved identifier denoting the predicate R of degree n. Then, a procedure of the type S(n) $R(x_1,x_2, \ldots ,x_n) \rightarrow S(x_1,x_2, \ldots ,x_n)$ we express as

$$S(x_1,x_2, \ldots ,x_n) \Leftrightarrow R(x_1,x_2, \ldots ,x_n) \text{ or more briefly as } S(n) \Leftrightarrow R(n)$$

The following types of expressions are admissible

(i) $S(n) \Leftrightarrow R(n)$

(ii) $S(n+1) \Leftrightarrow R(n)$

(iii) $T(n) \Leftrightarrow R(n) \wedge S(n)$ [$\wedge$ denotes the logical connective AND]

(iv) $T(n) \Leftrightarrow R(n) \vee S(n)$ [$\vee$ denotes the logical connective OR]

At times, one might come across a situation where a relation R denoted by reserved identifier R is of degree greater than that of its generated relation S by one In such a case, $S(n) \Leftrightarrow R(n+1)$ is not an admissible expression On the other hand, the projection S of R is expressed as $S(x_1,x_2,\ldots,x_n) \Leftrightarrow (\exists y) R(x_1,x_2,\ldots,x_n,y)$ which means that there is a y for which $R(x_1,x_2, \ldots x_n,y) \rightarrow S(x_1,x_2,\ldots,x_n)$ is valid

With these details, we now show that given an alphabet $\mathcal{A}$ and a normal algorithm over it, one can always construct a functionally equivalent EFS procedure in the language EFS(spl($\mathcal{A}$))

Let us consider the following EFS(spl($\mathcal{A}$)) procedure RRW(2), whose work space W' is the basic work space W of EFS(spl($\mathcal{A}$)) along with the reserved identifier FAC denoting the relation (*factor of*) FAC of rank 2. Let RRW(2) be a generated relation that represents the binary relation *rewritten string of* Now, we construct a procedure which will do the same job, that a substitution formula in a normal algorithm would do

RRW(2)

$SBT(\Lambda,x) \to RRW(\Lambda,x),$

$SBT(x,\Lambda) \to RRW(x,\Lambda),$

$FAC(Q,x) \to SBT(x,y) \to FAC(Q_1,y) \to RRW(Q,Q_1)$

A trace of RRW(2) for $\mathcal{A}$ = { 0 1 } would be of the following form

RRW(2)

$SBT(\Lambda, 101) \to RRW(\Lambda, 101),$

$SBT(101,\Lambda) \to RRW(101,\Lambda),$

$FAC(110110, 101) \to SBT(101,001) \to FAC(100110,001) \to RRW(110110, 100110)$

So, given an alphabet $\mathcal{A}$, one can realize any substitution formula over $\mathcal{A}$ with the help of the procedure RRW(2) In other words, for any normal algorithm $\mathcal{N}$ over $\mathcal{A}$, there is a functionally equivalent EFS(spl($\mathcal{A}$)) procedure

EXAMPLE 5.2 2 3

Let us construct a procedure for the cyclic permutation of a string of symbols from an alphabet $\mathcal{A}$

Language used	EFS(spl($\mathcal{A}$))
Data structure	$< \mathcal{A}^*, CON_{\mathcal{A}}, SBT_{\mathcal{A}} >$
Work space W″	Basic work space W, together with the reserved identifiers (i) FAC and (ii) RRW
Name of the procedure	CPS(2) [CPS represents the binary relation *Cyclically Permuted String of*]

CPS(2)

$CPS(\Lambda,\Lambda),$

$CPS(\xi, \xi),$

$CON(u,\xi,x) \to CON(\xi,u,x_1) \to SBT(x,x_1) \to RRW(x,x_1) \to$
$CPS(x,x_1).$

The fact that for a normal algorithm over an alphabet $\mathcal{A}$ there is a

functionally equivalent EFS(spl($\mathcal{A}$)) procedure, admits the possibility of obtaining EFS(spl($\mathcal{A}$)) procedures for constructive signal processing systems We shall verify this in the following

Firstly, we introduce here the notion of a *reserved alphabet* as an EFS analogue to the alphabet of auxiliary symbols By a reserved alphabet, we mean an alphabet that is included in the work space whose symbols do not appear in the output of a procedure.

Next, we show one method of implementing a binary operation between two discrete data sequences p(n) and q(n) by means of an EFS(spl($\mathcal{A}$)) procedure

As outlined in subsection 4.2, the data samples are coded as words from a suitable alphabet $\mathcal{A}$ so that the data sequences p(n) of M samples and q(n) of N samples could be expressed as the coded strings $P\uparrow d_p$ and $Q\uparrow d_q$ respectively, where $P\uparrow d_p = P_1 d_p P_2 d_p \ P_i d_p \ d_p P_M$, $1 \leq i \leq M$ and $Q\uparrow d_q = Q_1 d_q Q_2 d_q . Q_j d_q . d_q Q_N$, $1 \leq j \leq N$, d_p and d_q are two different delimiters from the delimiter alphabet $\mathfrak{D}$ Now, these coded strings are expressed as a $*$-pair $(P\uparrow d_p)*(Q\uparrow d_q)$ where the auxiliary symbol $*$ is a delimiter from the alphabet $\mathfrak{D}$ Then, the $*$-pair is manipulated by means of a suitable normal algorithmic system $\mathfrak{R}$ in such a way that the resulting string $R\uparrow d_r$ corresponds to the output r(n) due to the intended operation between the given data sequences p(n) and q(n).

A $*$-system of words from an alphabet $\mathcal{A}$ is an admissible term in an EFS procedure Let the symbol $*$ denote a binary operation Then, $*$-terms such as $(u*v)$ and $((u*v)*(w*x))$ are admissible atomic terms in an EFS procedure A sequence of atomic terms is called a *formation sequence* If in a formation sequence, the last term happens to be a $*$-term, then the $*$-term is called a *type-1 term*. Such type-1 terms are allowed in the procedure statements of EFS(spl($\mathcal{A}$)) also This means $*$-pair inputs to normal algorithms also act as inputs to EFS(spl($\mathcal{A}$)) procedures

As described in [14], an input accepting procedure would consist of the header

statement of the form shown below.

OUT_NAME (n) INPUT IN_NAME (k)
(Procedure Body of OUT_NAME)

OUT_NAME represents the name of a procedure and the identifier corresponding to the output which is an n-place relation IN_NAME represents the input to the procedure and it is a k-place relation IN_NAME is neither a reserved identifier nor a generated relation If the IN_NAME and OUT_NAME are simultaneously n-place relations, then the corresponding input accepting procedure is said to give rise to a *monotone operator* ϕ which is nothing but a mapping from a set of n- lace relations to another set of n-place relations satisfying the condition

$$X \subseteq Y \Rightarrow \phi(X) \subseteq \phi(Y)\ ,\quad X,Y \in D, \text{ the domain of the EFS}$$

We recall from subsection 4 2, that a constructive signal processing system is a normal algorithm or a set of normal algorithms combined in a manner that is allowed by any of the combination theorems of Markov

In the same way, procedures that are written in various work spaces of EFS(spl($\mathcal{A}$)), can be combined on the basis of the following theorems .

THEOREM 5 2 2 1

For any three relations $R(n_1)$, $S(n_2)$ and $T(n_3)$, if $S(n_2)$ is the generated relation of $R(n_1)$ being applied to a procedure P_i in W'_i and $T(n_3)$ is the generated relation of $S(n_2)$ being applied to a procedure P_j in W'_j , then one can construct a procedure P_k in W'_k which is the union of W_i and W_j such that $T(n_3)$ is a generated relation of $R(n_1)$ being applied to P_k

PROOF·

Let us assume that $n_1 = n_2 = n_3 = n$ Then ϕ_i , ϕ_j and ϕ_k can be treated as the monotone compact operators defined by the input accepting procedures P_i , P_j and P_k Now, as Fitting shows in [14], if the input to an operator defined by a procedure is a generated relation, then so is the output.

In the light of this proposition, we observe that if R is an input relation to ϕ_i

such that its output generated relation S when applied to ϕ_j yields the generated relation T. Then, there exisits a monotone operator ϕ_k which is the composition of ϕ_i and ϕ_j such that $\phi_j(\phi_i(R)) = T = \phi_k(R)$

The generalization of $\phi_i\phi_j = \phi_k$ to relations with $n_1 \neq n_2 \neq n_3$ yields $P_iP_j=P_k$ which is known as the composition of procedures

THEOREM 5 2 2 2

For any three relations $R(n_1)$, $S(n_2)$ and $T(n_3)$, if $S(n_2)$ is the generated relation of $R(n_1)$ being applied to a procedure P_i in W'_i and $T(n_3)$ is the generated relation of $R(n_1)$ being applied to a procedure P_j in W'_j, then one could construct a procedure P_k in W'_k so that the generated relation of $R(n_1)$ being applied to P_k in W'_k is the union of the relations $S(n_2)$ and $T(n_3)$

PROOF

This theorem is applicable to only those inputs which are applicable to both P_i and P_j Let us assume that $n_1 = n_2 = n_3 = n$ Then one can construct monotone operators corresponding to P_i, P_j and P_k which satisfy the union property such as

$$[\phi_i U \phi_j] (R) = \phi_i(R) U \phi_j(R) = \phi_k(R)$$

This theorem can also be generalized for input accepting procedures which generate relations of unequal arities

We now give three additional theorems concerning the combination of procedures, they can be proved on the same lines as theorem 5 2.2 2. Theorem 5.2 2 4 is a reformulation of a result of Fitting [14]

THEOREM 5 2.2.3

For any three relations $R(n_1)$, $S(n_2)$ and $T(n_3)$, if $S(n_2)$ is the generated relation of $R(n_1)$ being applied to a procedure P_i in W'_i and $T(n_3)$ is the generated relation of $R(n_1)$ being applied to a procedure P_j in W'_j, then one could construct a procedure P_k in W'_k so that the generated relation of $R(n_1)$ being applied to P_k in W'_k is the intersection of the relations $S(n_2)$ and $T(n_3)$

THEOREM 5 2 2 4

For any three procedures P_i , P_j and P_k in W'_i , W'_j and W'_k respectively, one can construct another procedure P_ℓ in W'_ℓ so that, (i) if P_k does not accept R then neither P_ℓ too, and (ii) if P_k accepts R then $P_\ell(R)$ is equivalent to $P_i(R)$ when $P_k(R)$ satifies a condition C and $P_\ell(R)$ is equivalent to $P_j(R)$ when $P_k(R)$ does not satisfy the condition C

THEOREM 5.2.2 5

For an input accepting procedu e P_i in W'_i one can construct another P_j in W'_j such that S is the generated relation of R being applied to P_i and (i) S is the least fixed point for P_j so that $P_j(S) = S$ and S is the output or (ii) S is again fed to P_i and the corresponding generated relation is tested for the least fixed point P_i is repeatedly applied till its output becomes the least fixed point of P_j.

REMARKS

(i) The notion of a type-i term allows us to feed a ✱-system of strings from the basic alphabet as the input to an input accepting EFS(spl($\mathcal{A}$)) procedure

(ii) The notion of a reserved alphabet allows auxiliary symbols in the statements of an EFS(spl($\mathcal{A}$)) procedure.

(iii) Complex EFS(spl($\mathcal{A}$)) procedures can be constructed by making use of the above combination theorems

Finally we remark here, that the string manipulation language EFS(spl($\mathcal{A}$)) is a useful computational aid in realizing constructive signal processing operations.

5 3 LANGUAGES GENERATED BY M-GRAMMAR BASED ON ON SEMI-THUE PRODUCTIONS OF POST WORDS

So far we explored the possibilities of studying constructive signal processing systems in terms of formal languages Using the concept of *Elementary Formal System* of Smullyan, we proposed a string manipulation language EFS(spl($\mathcal{A}$))

as an extension to that of Fitting's EFS(str(L)). Finally we gave a method of implementing signal processing operations by means of certain EFS(spl($\mathcal{A}$)) procedures.

In this subsection, we consider the problem of finitely specifying the potentially infinite languages of a free monoid recognized by constructive signal processing systems, by means of a set of rules called *M-grammar*.

5 3 1 THE PROBLEM OF GENERATING MARKOV CLASS REWRITING SYSTEMS BY A TYPE-0 PHRASE STRUCTURE GRAMMAR

We start with the following definition

DEFINITION 5.3 1 1

A *Markov class rewriting system* (*process*) is defined as an ordered pair $<\Sigma, S>$ where the alphabet Σ is the union of the *terminal* (basic) alphabet $\mathcal{A}$ and the alphabet $\mathcal{B}$ consisting of *non terminals* (auxiliary variables) , and S is a totally ordered list of semi-Thue productions, which are pattern substitution formulas of the type any string of $\Sigma^* \longrightarrow$ any string of Σ^*

With this definition in mind, we shall view a normal algorithm as a Markov class rewriting process, or more generally, as a Markov class rewriting system

Now, the following two theorems of the established literature highlight the fact that Markov class rewriting systems can be defined only by a type-0 grammar of Chomsky heirarchy

THEOREM 5 3 1 1 (Detlovs) [21]

The class of normally computable partial word functions is equivalent to the class of partial recursive word functions

THEOREM 5 3 1.2 (Davis and Chomsky) [54]

A language $L \subseteq \mathcal{A}^*$ is generated by a formal grammar $<\Sigma,P,\mathcal{A}>$ if and only if it is recursively enumerable

The languages recognized by the class of normal algorithms over an alphabet $\mathcal{A}$ are recursively enumerable and so can be finitely specified only by a type-0 grammar

DEFINITION 5 3 1 2 [54]

A (type-0) phrase structure grammar G_T , also called a semi-Thue process, is an ordered quadruple $\langle\Sigma,P,S,\mathcal{A}\rangle$ where,

(i) Σ is a total alphabet,

(ii) $\mathcal{A}$ is the terminal alphabet,

(iii) $\Sigma\backslash\mathcal{A}$ is the alphabet of non terminals,

(iv) P is a set of semi-Thue rewriting rules of the type String 1 $\longrightarrow$ String 2 where String 1 can be any string of terminals and non terminals that contains at least one non terminal and String 2 is any string of terminals and non terminals whatsoever, and ,

(v) S is known as a *Start symbol* which is in $\Sigma\backslash\mathcal{A}$.

G_T defines a semi-Thue rewriting system $\langle\Sigma,P\rangle$.

From the above definition it is clear that a type-0 grammar does not allow semi-Thue productions of the form $\Lambda \longrightarrow$ String 2 where String 2 is from Σ. But the notion of a normal algorithm does allow substitution formulas of the form $\Lambda \longrightarrow$ String 2 Therefore, it is not possible to define Markov class rewriting systems directly by a type-0 grammar of Chomsky heirarchy

We overcome this difficulty by introducing a grammar, which we call M-grammar, for describing Markov class rewriting systems To do this, we restructure substitution formulas as semi-Thue productions corresponding to certain representations of a Turing machine, called *quadruples* that operate on special words known as Post words Post words are words of the type $hkq_w\ell h$ where h is an auxiliary letter, k and ℓ are words from an alphabet S and q_w is a letter from a state alphabet of a Turing machine

5 3 2 FORMULATION OF M-GRAMMAR

Recently Babikov [36] proposed a method for realizing normal algorithms in terms of Turing machines by representing normal algorithms in an index alphabet of a semi-Thue system Using this method, every substitution of a normal algorithm is replaced by a certain system of substitutions of a semi-Thue system which operates on Post words and thus the appropriate Turing machine is determined Babikov refers to Post words as words of the type $hkq_w\ell h$ where h is an auxiliary letter, k and ℓ are words from the alphabet $\mathbb{C}\cup\{\Lambda\}$ over which the normal algorithm is constructed, q is a letter from Q and w is a word from $S\cup\{1,2,3, \quad ,m\}$; $m=f(n)$ is a finite integer function of n. Our M-grammar is primarily based on this idea We proceed as follows

Recall that a Turing machine is described by a set of quadruples of the following types ·

(i) $q_i s_j s_k q_\ell$ (being in the state q_i the machine scans the symbol s_j , prints s_k in the place of s_j and goes into state q_ℓ)

(ii) $q_i s_j R q_\ell$ (being in the state q_i the machine scans the symbol s_j, moves to the right adjacent square and goes into state q_ℓ)

(iii) $q_i s_j L q_\ell$ (being in the state q_i the machine scans s_j , moves to the left adjacent square and goes into state q_ℓ)

For deterministic Turing machines, no two quadruples can have the same pair $q_i s_j$

Now, let us consider a deterministic Turing machine $\mathcal{M}$ with the alphabet $S = \{s_1,s_2, \quad ,s_k \}$ and the state alphabet $Q = \{ q_1,q_2,..,q_n \}$ The semi-Thue productions corresponding to the quadruples of this Turing machine are as follows

(i) $q_i s_j s_k q_\ell$ corresponds to the semi Thue production $q_i s_j \longrightarrow q_\ell s_k$

(ii) $q_i s_j R q_i$ corresponds to 1. $q_i s_j s_k \longrightarrow s_j q_i s_k$, ($k=0,1,2, \quad ,K$)

$$2 \quad q_i s_j h \longrightarrow s_j q_1 \Lambda h$$

(iii) $\quad q_i s_j L q_1$ corresponds to $\cdot$ $1 \quad s_k q_i s_j \longrightarrow q_1 s_k s_j$, ($k=0,1,2, \ldots,K$)

$$2 \quad h q_i s_j \longrightarrow h q_1 \Lambda s_j$$

Let us assume the configuration of $\mathcal{M}$ at a particular stage of computation to be

$$\begin{array}{cccc} s_2 & s_1 & s_0 & s_3 \\ & \uparrow & & \\ & q_4 & & \end{array}$$

This corresponds to the Post word $hs_2q_4s_1s_0s_3h$ Let us assume that $\mathcal{M}$ contains the quadruple $q_4s_1s_3q_5$ so that the corresponding semi Thue production is of the form $q_4s_1 \longrightarrow q_5s_3$ According to this production rule, the Post word $hs_2q_4s_1s_0s_3h$ is rewritten as $hs_2q_5s_3s_0s_3h$ In a similar manner, the rewriting of Post words by virtue of the production rules corresponding to the remaining two types of quadruples of $\mathcal{M}$ can be verified. A system of semi Thue productions corresponding to certain quadruples of a Turing machine $\mathcal{M}$ and which operate on Post words, is called a *semi-Thue system* denoted by $\Sigma(\mathcal{M})$

All the three types of semi Thue production rules allow only *letter to letter substitution* But a *word to word substitution formula* of a normal algorithm, as we know, is a restriction-free mapping of the type $\Sigma^* \longrightarrow \Sigma^*$ So the question that arises here is whether it is possible to restructure the substitution formulas of a normal algorithm as semi-Thue production rules which operate on Post words.

In this context, we propose the following technique by which a *word to word substitution* is realized by a *block of letter to letter substitutions* Firstly we shall agree to the following assumption .

ASSUMPTION

Any configuration of a Post word can be represented in infinite number of ways by including any number of *blanks* corresponding to null strings, in between any two symbols in the configuration

On the basis of this assumption, we can make the lengths of the pair of words

that form a substitution formula, equal, by padding the shorter word with the required number of Λ's For example, we can restructure the substitutiton formulas (i) $\beta\xi \longrightarrow \alpha\xi\beta$ and (ii) $\Box * \xi \longrightarrow \Box *$ as (i) $\beta\xi\Lambda \longrightarrow \alpha\xi\beta$ and (ii) $\Box * \xi \longrightarrow \Box * \Lambda$ respectively

For convenience, we shall call this technique *End Justification*. End justification is done in two ways, that is, the pair of words of a substitution formula is either left justified by suffixing the shorter word with Λ's or right justified by prefixing the shorter word with Λ's After the end justification of the pair of words, the word to word substitution due to the corresponding substitution formula which is a map of the type $\Sigma^* \longrightarrow \Sigma^*$ could be replaced by a block of letter to letter substitutions of the type $\Sigma \cup \{\Lambda\} \longrightarrow \Sigma \cup \{\Lambda\}$ Now, the block of semi-Thue production rules corresponding to the block of letter to letter substitutions is obtained in the manner described above

We illustrate this technique of restructuring a substitution formula in the following example

EXAMPLE 5 3 2 1

Let us consider a substitution formula $s_1s_2s_1s_3 \longrightarrow s_2s_2s_1$ Then, the end justified version of this formula would be of the form $s_1s_2s_1s_3 \longrightarrow s_2s_2s_1\Lambda$ and the corresponding block of semi-Thue productions would contain the following ·

Table 5 3 2.1 Semi-Thue productions corresponding to the end justified substitution formula $s_1s_2s_1s_3 \longrightarrow s_2s_2s_1\Lambda$

Sl No	Post words	Semi-Thue productions	Remarks
1	$q_{10}s_1s_2q_{11}$	$q_{10}s_1 \longrightarrow q_{11}s_2$	letter substitution
2	$q_{11}s_2Rq_{11}$	$q_{11}s_2s_2 \longrightarrow s_2q_{11}s_2$	right move

contd..

3	$q_{11}s_2s_2q_{12}$	$q_{11}s_2 \longrightarrow q_{12}s_2$	letter substitution
4	$q_{12}s_2Rq_{12}$	$q_{12}s_2s_1 \longrightarrow s_2q_{12}s_1$	right move
5	$q_{12}s_1s_1q_{13}$	$q_{12}s_1 \longrightarrow q_{13}s_1$	letter substitution
6	$q_{13}s_1Rq_{13}$	$q_{13}s_1s_3 \longrightarrow s_1q_{13}s_3$	right move
7	$q_{13}s_3\Lambda q_{14}$	$q_{13}s_3 \longrightarrow q_{14}\Lambda$	letter substitution

Now, the semi-Thue system $\Sigma(\mathcal{M})$ consisting of the above production rules operates on the corresponding Post words, thus simulating the effect of the formula $s_1s_2s_1s_3 \xrightarrow[\Sigma(\mathcal{M})]{} s_2s_2s_1$.

Let us consider a normal algorithm $\mathcal{N}$ over an alphabet $\mathcal{A}$ whose scheme consists of n substitution formulas Then, each of the n formulas can be replaced by the appropriate blocks of semi-Thue rewriting rules The totally ordered list of all such blocks gives rise to what we call as *End-justified Post Turing* (EPT) *rewriting system* whose definition is given below

DEFINITION 5 3 2 1

An EPT rewriting system is an RE-analogue (Recursively Enumerable) of a Markov class rewriting system and is defined as the 4-tuple $\langle\Sigma,Q,h,P\rangle$ where Σ is the total alphabet, Q is the state alphabet consisting of { q_{ij} $0\leq i\leq m$, $0\leq j\leq n$ such that q_{ij} represents states corresponding to the i^{th} block of end justified words of length n }, h is an auxiliary symbol known as *end marker* and which is used in the construction of Post words and P is the subset of the Cartesian product $(h\Sigma^*q_{ij}\Sigma^*h) \times (h\Sigma^*q_{kl}\Sigma^*h)$, known as the finite list of semi-Thue productions which operate on Post words

We shall demonstrate the working principle of an EPT rewriting system by means of the following example

EXAMPLE 5 3 2.2

Let us consider the scheme of the cyclic shifting normal algorithm, $\mathcal{N}^{cs}$ [Ref Subsection 4 1 1] The semi-Thue system corresponding to $\mathcal{N}^{cs}$ is the totally ordered list of *ten blocks* of semi-Thue rewriting rules as given in table 5 3 2 2

Table 5 3 2 2 Semi-Thue system corresponding to $\mathcal{N}^{cs}$

Sl No	Semi-Thue productions	Remarks
0	$q_{INI}\Sigma \longrightarrow q_{00}\Sigma$	Input
1	$q_{00}\backslash\xi \longrightarrow q_{10}\Sigma\backslash\xi$	
2	$q_{00}\xi \longrightarrow q_{01}\mu$	
3	$q_{01}\mu\Sigma \longrightarrow \mu q_{02}\Sigma$	
4	$q_{02}\Sigma\backslash\alpha \longrightarrow q_{10}\Sigma\backslash\alpha$	
5	$q_{02}\alpha \longrightarrow q_{03}\Gamma$	Block 0 corresponding to the formula $\xi\alpha\mu\Gamma \longrightarrow \mu\Gamma\xi\alpha$
6	$q_{03}\Gamma\Sigma \longrightarrow \Gamma q_{04}\Sigma$	
7	$q_{04}\Sigma\backslash\mu \longrightarrow q_{10}\Sigma\backslash\mu$	
8	$q_{04}\mu \longrightarrow q_{05}\xi$	
9	$q_{05}\xi\Sigma \longrightarrow \xi q_{06}\Sigma$	
10	$q_{06}\Sigma\backslash\Gamma \longrightarrow q_{10}\Sigma\backslash\Gamma$	
11	$q_{06}\Gamma \longrightarrow q_{07}\alpha$	
12	$q_{07}\alpha \longrightarrow \alpha q_{08}\Lambda$	

contd

13	$q_{08}\Lambda \longrightarrow q_{00}\Lambda$	
14	$q_{10}\Sigma\backslash\alpha \longrightarrow q_{20}\Sigma\backslash\alpha$	Block 1 corresponding to the formula $\alpha\mu\Gamma \longrightarrow \mu\Gamma\alpha$
15	$q_{10}\alpha \longrightarrow q_{11}\mu$	
16	$q_{11}\mu\Sigma \longrightarrow \mu q_{12}\Sigma$	
17	$q_{12}\Sigma\backslash\mu \longrightarrow q_{20}\Sigma\backslash\mu$	
18	$q_{12}\mu \longrightarrow q_{13}\Gamma$	
19	$q_{13}\Gamma\Sigma \longrightarrow \Gamma q_{14}\Sigma$	
20	$q_{14}\Sigma\backslash\Gamma \longrightarrow q_{20}\Sigma\backslash\Gamma$	
21	$q_{14}\Gamma \longrightarrow q_{15}\alpha$	
22	$q_{15}\alpha \longrightarrow \alpha q_{16}\Lambda$	
23	$q_{16}\Lambda \longrightarrow q_{00}\Lambda$	
24	$q_{20}\Sigma\backslash\beta \longrightarrow q_{30}\Sigma\backslash\beta$	Block 2 corresponding to the formula $\beta\mu \longrightarrow \alpha\mu\beta$
25	$q_{20}\beta \longrightarrow q_{21}\alpha$	
26	$q_{21}\alpha\Sigma \longrightarrow \alpha q_{22}\Sigma$	
27	$q_{22}\Sigma\backslash\mu \longrightarrow q_{30}\Sigma\backslash\mu$	
28	$q_{22}\mu \longrightarrow q_{23}\mu$	
29	$q_{23}\mu\Lambda \longrightarrow \mu q_{24}\Lambda$	
30	$q_{24}\Lambda \longrightarrow q_{25}\beta$	

contd

31	$q_{25}\beta \longrightarrow \beta q_{26}\Lambda$	
32	$q_{26}\Lambda \longrightarrow q_{00}\Lambda$	
33	$q_{30}\Sigma\backslash\alpha \longrightarrow q_{40}\Sigma\backslash\alpha$	
34	$q_{30}\alpha \longrightarrow q_{31}\beta$	
35	$q_{31}\beta\Sigma \longrightarrow \beta q_{32}\Sigma$	Block 3 corresponding to the formula $\alpha\alpha \longrightarrow \beta$
36	$q_{32}\Sigma\backslash\alpha \longrightarrow q_{40}\Sigma\backslash\alpha$	
37	$q_{32}\alpha \longrightarrow q_{33}\Lambda$	
38	$q_{33}\Lambda \longrightarrow q_{00}\Lambda$	
39	$q_{40}\Sigma\backslash\beta \longrightarrow q_{50}\Sigma\backslash\beta$	
40	$q_{40}\beta \longrightarrow q_{41}\Gamma$	
41	$q_{41}\Gamma\Sigma \longrightarrow \Gamma q_{42}\Sigma$	Block 4 corresponding to the formula $\beta\beta \longrightarrow \Gamma$
42	$q_{42}\Sigma\backslash\beta \longrightarrow q_{50}\Sigma\backslash\beta$	
43	$q_{42}\beta \longrightarrow q_{43}\Lambda$	
44	$q_{43}\Lambda \longrightarrow q_{00}\Lambda$	
45	$q_{50}\Sigma\backslash\Gamma \longrightarrow q_{60}\Sigma\backslash\Gamma$	
46	$q_{50}\Gamma \longrightarrow q_{51}\Gamma$	
47	$q_{51}\Gamma\Sigma \longrightarrow \Gamma q_{52}\Sigma$	Block 5 corresponding to the formula $\Gamma\alpha \longrightarrow \Gamma$
48	$q_{52}\Sigma\backslash\alpha \longrightarrow q_{60}\Sigma\backslash\alpha$	

contd..

49	$q_{52}\alpha \longrightarrow q_{53}\Lambda$	
50	$q_{53}\Lambda \longrightarrow q_{00}\Lambda$	
51	$q_{60}\Sigma\backslash\Gamma \longrightarrow q_{70}\Sigma\backslash\Gamma$	Block 6 corresponding to the formula $\Gamma\mu \longrightarrow \mu\Gamma$
52	$q_{60}\Gamma \longrightarrow q_{61}\mu$	
53	$q_{61}\mu\Sigma \longrightarrow \mu q_{62}\Sigma$	
54	$q_{62}\Sigma\backslash\mu \longrightarrow q_{70}\Sigma\backslash\mu$	
55	$q_{62}\mu \longrightarrow q_{63}\Gamma$	
56	$q_{63}\Gamma \longrightarrow q_{64}\Lambda$	
57	$q_{64}\Lambda \longrightarrow q_{00}\Lambda$	
58	$q_{70}\Sigma\backslash\beta \longrightarrow q_{80}\Sigma\backslash\beta$	Block 7 corresponding to the formula $\beta\alpha \longrightarrow \beta$
59	$q_{70}\beta \longrightarrow q_{71}\beta$	
60	$q_{71}\beta\Sigma \longrightarrow \beta q_{72}\Sigma$	
61	$q_{72}\Sigma\backslash\alpha \longrightarrow q_{80}\Sigma\backslash\alpha$	
62	$q_{72}\alpha \longrightarrow q_{73}\Lambda$	
63	$q_{73}\Lambda \longrightarrow q_{00}\Lambda$	
64	$q_{80}\Sigma\backslash\Gamma \longrightarrow q_{90}\Sigma\backslash\Gamma$	Block 8 correspon--ding to the formula $\Gamma \longrightarrow \cdot$
65	$q_{80}\Gamma \longrightarrow q_{FIN}\Lambda$	
66	$q_{90}\Lambda \longrightarrow q_{91}\alpha$	Block 9 corresponding the formula

contd

67	$q_{91}\alpha \longrightarrow \alpha q_{92}\Lambda$	$\longrightarrow \alpha$
68	$q_{92}\Lambda \longrightarrow q_{00}\Lambda$	

Now, we define M-grammar

DEFINITION 5 3 2 2

The M-grammar is defined as a 7-tuple $G_M = < \Sigma, Q, INI, FIN, h, \Psi, P >$ where $\Sigma = S \cup \{\Lambda\}$ is the total alphabet consisting of the terminal and the non terminal alphabets and the null string Λ , Q is the state alphabet, the number of eleme :s of which depends on the scheme of a normal algorithm , $INI \notin Q$ is the initial state and $FIN \notin Q$ is the final state such that $Q' = Q \cup \{INI\} \cup \{FIN\}$, h is the end marker of a Post word from $\Sigma \cup Q' \cup \{h\}$, Ψ is the start axiom from $\Sigma \cup Q' \cup \{h\}$ and P is the totally ordered list of semi-Thue productions operating on Post words

M-grammar defines EPT rewriting systems The direct derivation relation of an EPT system is denoted by $\Rightarrow$ and defined as follows

For some Post words u and v from $\Sigma \cup Q' \cup \{h\}$, $u \underset{G_M}{\Rightarrow} v$ means that there are certain $x_1, x_2, x_3 \in \Sigma^*$ and $q_{ij}, q_{k\ell} \in Q'$ such that $< u, v > \in P$, that is, $< (hx_1q_{ij}x_2h), (hx_1q_{k\ell}x_3h) > \in P$ and $(q_{ij}x_2x_3q_{k\ell})$ is a quadruple of a Turing machine

The language generated by G_M is defined as $L(G_M)$ where,

$$L(G_M) = \{ w \in h\mathcal{A}^*Q'\mathcal{A}^*h \quad \Psi \in h\Sigma^*Q'\Sigma^*h \overset{*}{\underset{G_M}{\Rightarrow}} w \}$$

G_M generates potentially infinite languages of an alphabet and such a family of languages is denoted by L(M). Now the following theorem ascertains the recursive enumerability of the languages generated by G_M

THEOREM 5 3.2 1

L(M) = L(RE) (The acronym RE stands for Recursively Enumerable)

The proof of this theorem is the direct consequence of the theorem 5 3 1 2

Finally we remark here, that a language recognized by a constructive signal processing system is a subset of L(M)

SECTION 6

SPECIAL AUTOMATA FOR NORMAL ALGORITHMS AND THEIR TRANSCRIPTIONS

It was shown in section 5, that symbolic signal processing operations could be carried out by means of Markov class EPT rewriting systems that are described by M-grammar

It was also mentioned in the beginning of section ᒪ, that a language is a subset of a free monoid and is recognized by a finite state machine (automaton) A language L is recognized by an automaton in such a manner that if we start with its initial state and feed a string from the free monoid, the final state will belong to a designated set if and only if the string belongs to L.

The notions of automata and codes are closely associated with each other, in the sense that, a *subset of a free monoid* is not only known as a *language* but also as a *code* and the finite state machine that recognizes it, is called an automaton. Any signal representation is a code by itself and any finite state machine that simulates a signal processing system is an automaton

In this section, we shall interpret signals as variable length codes over alphabets and normal algorithms as special automata, and briefly demonstrate a syntactic method of implementing signal processing operations by means of a system of such automata of appropriate normal algorithms

There are occasions when it becomes necessary to encode the schemes of certain normal algorithms such that their coded forms become amenable to some other normal algorithms Hence, we shall treat a constructive signal processing system either as a *code* or as an *automaton* depending on the requirement. Before going into the details, we shall review some of the preliminaries required for our study from [6] , [21], [22] and [26] As tutorial aids, examples relevant to our study

are added to clarify the main ideas

6 1 CODES AND AUTOMATA

Let $\mathcal{A}$ be an alphabet and $\mathcal{A}^*$ be its free monoid Then, a word $w \in \mathcal{A}^*$ is called *primitive* if it is not a power of any other word For example, the word P = 10101 from the alphabet $\mathcal{A}_0$ = { 0 1 } is a primitive word whereas the word Q = 1010 is not, because Q is the second power of the word 10

Two words x and y are said to be *conjugate* if there exist two other words u and v such that x = uv and y = vu Conjugacy relation is an equivalence relation Equivalence classes for this relation are called *conjugacy classes* It is at times convenient to say of conjugate words x and y that x is a conjugate of y or equivalently y is a conjugate of x [Note The null string Λ is a conjugate of itself]

Given a pair of conjugate words, one can obtain a word of the pair from the other with the help of the cyclic shifting normal algorithm $\mathcal{N}^{cs}$ [Subsection 4.1 1]. For example, x = 010101 and y = 101010 are two conjugate words from the alphabet $\mathcal{A}_0$ = { 0 1 } such that $x = \mathcal{N}^{cs}(y)$ and $y = \mathcal{N}^{cs}(x)$ Likewise, in the case an arbitrary pair of conjugate words, x and y, we can get x from y by n applications of $\mathcal{N}^{cs}$, $x = (\mathcal{N}^{cs})^n(y)$, for some n, $0 \le n \le |y|$

[Note. $|y|$ denotes the length of the word y and $(\mathcal{N}^{cs})^n(y)$ denotes the n number of successive applications of the normal algorithm $\mathcal{N}^{cs}$ to the word y]

PROPOSITION 6 1 1

Let C be a conjugacy class of words of length ℓ with an exponent n Let x be a word in C of the form $x = r^n$ where the primitive word r is a root Then, for any other word y in C there is a root q such that $y = q^n$ Further, the cardinality of C is ℓ/n

PROOF

Let x be a word in C such that $x = r^n$ and let $|r|$ be p Then $|x| = pn = \ell$. Now, the normal algorithm $\mathcal{N}^{cs}$ could be applied successively to the word x up to a maximum of p-1 times without affecting the condition that all words of C have the same exponent n In that case the conjugacy class C would consist of the following p words x, $\mathcal{N}^{cs}(x)$, $(\mathcal{N}^{cs})^2(x)$, $(\mathcal{N}^{cs})^3(x)$, , $(\mathcal{N}^{cs})^{p-1}(x)$ where $p = \ell/n$.

EXAMPLE 6 1 1

Let us consider the alphabet $\mathcal{A} = \{ a\ b\ c \}$, a conjugacy class $C = \{ x \mid x$ is a word of length 12 and exponent 4 $\}$ and a word x = abcabcabcabc in C where r = abc and n = 4 Now, $\mathcal{N}^{cs}(x)$ = cabcabcabcab where r = cab and n = 4 and $(\mathcal{N}^{cs})^2(x)$ = bcabcabcabca where r = bca and n = 4 So, the cardinality of C is 3

DEFINITION 6 1 1

A subset of X of $\mathcal{A}^*$ is called a *code over the alphabet* $\mathcal{A}$ if for all $n, m \geq 1$ and $x_1x_2x_3\ x_n$, $x'_1x'_2x'_3\ x'_m \in X$, the condition $x_1x_2x_3\ x_n = x'_1x'_2x'_3\ x'_m$ implies $n = m$ and $x_i = x'_i$ for $i = 1,2,3, ,n$

Any subset of a code is also a code If $X \subset \mathcal{A}^*$ is a code, then X^n is also a code for all $n > 0$ In general, coding is defined as an injective morphism β , $\beta\ \mathcal{B}^* \longrightarrow \mathcal{A}^*$ for alphabets $\mathcal{A}$ and $\mathcal{B}$, such that the following conditions hold

(i) For every symbol $\xi \in \mathcal{B}$ there is one and only one $\beta(\xi)$ in $\mathcal{A}^*$

(ii) There is a proper subset X of $\mathcal{A}^*$ such that β induces a bijection of $\mathcal{B}$ onto X

The family of all possible codings from $\mathcal{B}$ onto X is denoted by $COD(\mathcal{B},X)$ Let $\beta\ \mathcal{B}^* \longrightarrow \mathcal{A}^*$ be an injective morphism Then, $\beta(\mathcal{B}) = X$, where X is a proper subset of the free monoid $\mathcal{A}^*$ is known as the *coding morphism* for X

DEFINITION 6 1.2 .

The subset $X \subset \mathcal{A}^*$ is a *prefix (suffix) code* if no element of X is a proper left (right) factor of another element in X , that is, $X\mathcal{A}^+ \cap X = \phi$ for prefix code

($\mathcal{A}^+X \cap X = \phi$ for suffix code) The code $X \subset \mathcal{A}^*$ is known as *biprefix* if X is both prefix and suffix

Every submonoid M of a free monoid $\mathcal{A}^*$ has a unique minimal set of generators $X = (M-\Lambda)-(M-\Lambda)^2$ where Λ is the null string and X is a code.

DEFINITION 6 1 3

Let $\mathcal{A}$ be an alphabet Then, an *automaton over* $\mathcal{A}$ is defined as a triple $\mathbf{A} = \langle Q,I,T \rangle$ where Q is a set of states, I and T are subsets of Q and are respectively known as the *initial* and *final* states

Edge is an element from the 3-tuple $Q \times \mathcal{A} \times Q$ and the set of all edges of an automaton is denoted by $\mathcal{F} \subseteq Q \times \mathcal{A} \times Q$ A path of length n in an automaton is a sequence $c = (f_1, f_2, \ldots, f_n)$ of consecutive edges, where $f_i = (q_i, a_i, q_{i+1})$, $1 \le i \le n$ The label of a path c of length n is the word $w = a_1 a_2 a_3 \ldots a_n$ where $c = (f_1, f_2, \ldots, f_n)$ and $f_1 = (q_1, a_1, q_2)$, $f_2 = (q_2, a_2, q_3)$, $f_3 = (q_3, a_3, q_4)$, .. , $f_n = (q_n, a_n, q_{n+1})$ In short, a path c of length n with the label w is denoted by $c\ \ q_1 \xrightarrow{w} q_{n+1}$.

A path $c\ \ q_1 \longrightarrow q_t$ is called *successful* if and only if $q_1 \in I$ and $q_t \in T$ The set of all sucessful paths of an automaton $\mathbf{A}$ is denoted by $L(\mathbf{A})$ A state $q \in Q$ is *accessible* or *co-accessible* if there exists accordingly a path c which is either $c\ \ i \longrightarrow q$ and $i \in I$ or $c\ \ q \longrightarrow t$ and $t \in T$ An automaton whose every state is both accessible and co-accessible, is known as a *trim automaton* $\mathbf{A}' = \langle P, I \cap P, T \cap P \rangle$ where P is a set of states which are accessible and co-accessible Then, it can be verified that $L(\mathbf{A}) = L(\mathbf{A}')$

An automaton $\mathbf{A} = \langle Q,I,T \rangle$ is *deterministic* if the cardinality of I is 1 and if for any $p \in Q$ the edges (p,a,q) and (p,a,r) belong to the set $\mathcal{F}$ implies $q = r$. For every $p \in Q$ and $a \in \mathcal{A}$, the transition function for an automaton $\mathbf{A}$ is defined as :

$$p.a = q \quad \text{if } (p,a,q) \in \mathcal{F}$$
$$= \phi \quad \text{otherwise}$$

This is a partial function of the type $Q \times \mathcal{A} \longrightarrow Q$ An automaton is *complete* if

for every $p \in Q$ and every $a \in \mathcal{A}$ there exists at least one $q \in Q$ so that $(p,a,q) \in \mathfrak{F}$.

THEOREM 6 1 1

Let us consider an alphabet $\mathcal{A}$ and a subset X of the free monoid $\mathcal{A}^*$. Then X is recognized by a finite automaton

PROOF ·

Let ϕ $\mathcal{A}^* \longrightarrow M$ be a morphism onto a finite monoid M and let us assume that ϕ recognizes X Then, a finite automaton can be defined as $\mathbf{A} = \langle M, \Lambda, \phi(X) \rangle$ with the transition function $m\,a = m\phi(a)$ Let w be a word in X Ther $\Lambda\, w \in \phi(X)$ if and only if $\phi(w) \in \phi(X)$ Then $L(\mathbf{A}) = X$ Hence, X is recognized by $\mathbf{A}$

DEFINITION 6.1 4

An *asynchronous automaton* is a generalized notion of an automaton. It has edges labelled either by a letter or by the null string such that $\mathfrak{F} \subset Q \times (\mathcal{A} \cup \Lambda) \times Q$

THEOREM 6 1 2

If $X \subset \mathcal{A}^*$ is recognizable by an automaton $\mathbf{A}$, then X^* is also recognizable by $\mathbf{A}$ If $X, Y \subset \mathcal{A}^*$ are recognizable by $\mathbf{A}$, then XY is also recognizable by $\mathbf{A}$

PROOF .

Now $X \subset \mathcal{A}^*$ Since $X^* = (X-\Lambda)^*$, $\Lambda \notin X$ is a valid assumption Let $\mathbf{A} = \langle Q, I, T \rangle$ be a finite automaton recognizing X and let $\mathfrak{F}$ be the set of its edges Since $\Lambda \notin X$, edges of the types $q_1 \xrightarrow{\Lambda} q_1$ or $q_1 \xrightarrow{\Lambda} q_j$, q_1 , $q_j \in Q$ are not in $\mathfrak{F}$ This means $I \cap T = \phi$ Now let us construct another asynchronous automaton $\mathbf{B} = \langle Q, I, T \rangle$ whose set of edges is $\mathfrak{F}'$ where $\mathfrak{F}' = \mathfrak{F} \cup (T \times \{\Lambda\} \times I)$ In the light of the definition of a successful path, we observe that the number of successful paths in $\mathbf{B}$ amounts to the free semigroup X^+ This means $L(\mathbf{B}) = X^+$. Any successful path in $\mathbf{B}$ may contain an edge of the form c· $i_k \xrightarrow{w_k} t_k$, $t_k \in T$, $i_k \in I$ and an edge of the form c· $t_k \xrightarrow{\Lambda} i_k$ In other words, any word w in X^+ can be recognized as a successful path in $\mathbf{B}$ such as

c $i_1 \xrightarrow{w_1} t_1 \xrightarrow{\Lambda} i_2 \xrightarrow{w_2} t_2 \xrightarrow{\Lambda} i_3 \xrightarrow{w_3} . \quad \xrightarrow{\Lambda} i_n \xrightarrow{w_n} t_n$ and

$i_1 = i$ and $t_n = j$, $i \in I$, $j \in T$.

Since X^+ is recognizable, X^* is also recognizable if we assume that $I \cap T \neq \phi$ that is, $\Lambda \in X$

Now let us consider two finite automatons $\mathbf{A}_1 = \langle P,I,S \rangle$ and $\mathbf{A}_2 = \langle Q,I,T \rangle$ with sets of edges $\mathfrak{F}_1$ and $\mathfrak{F}_2$ respectively We shall construct an asynchronous automaton $\mathbf{A}_3$ with the set of edges $\mathfrak{F}_3$ where $\mathfrak{F}_3 = \mathfrak{F}_1 \cup \mathfrak{F}_2 \cup (S \times \{\Lambda\} \times J)$ Now, $\mathbf{A}_3$ recognizes XY

DEFINITION 6 1 5

To each automaton $\mathbf{A} = \langle Q,I,T \rangle$ with the set of edges $\mathfrak{F}$ over an alphabet $\mathcal{A}$, a function ϕ $\mathbf{A} \longrightarrow N^{Q \times Q}$ is associated which is defined as :

$$(p, \phi(a), q) = 1 \text{ if } (p,a,q) \in \mathfrak{F}$$

$$= 0 \text{ otherwise}$$

N is the semiring of natural numbers $N^{Q \times Q}$ is called the monoid of N relations over Q , that is, to each pair $(q_i , q_j) \in Q \times Q$ is related a number from N by virtue of the function ϕ.

ϕ is the morphism of the type $\mathcal{A}^* \longrightarrow N^{Q \times Q}$ if the following conditions hold

(i) $\phi(\Lambda) = I_Q$; (I_Q is the identity relation $(q_i , q_i) \in N^{Q \times Q}$)

(ii) For any $u, v \in \mathcal{A}^*$, $(q_i , \phi(uv) , q_j) = \sum_{q_k \in Q} (q_i , \phi(u) , q_k)(q_k , \phi(v) , q_j)$

Then the morphism ϕ is known as the *representation* of the automaton $\mathbf{A}$ over N.

DEFINITION 6 1 6

The formal power series, otherwise known as the *behaviour* of an automaton $\mathbf{A} = \langle Q,I,T \rangle$ over an alphabet $\mathcal{A}$ is defined by

$$(|\mathbf{A}| , w) = \sum_{i=1,\ t=1} (i, \phi(w), t)$$

and designated by $|\mathbf{A}|$ The support of $|\mathbf{A}|$ could be seen to be the set recognized by $\mathbf{A}$, i.e , the set $L(\mathbf{A})$

DEFINITION 6.1.7

To every automaton $\mathbb{A} = \langle Q,I,T \rangle$ over $\mathcal{A}$, there corresponds another automaton called $\mathbb{A}^*$, whose construction is as explained below.

Let us consider a new state q_* which is not in Q and construct an automaton $\mathbb{B} = \langle Q \cup q_*, q_*, q_* \rangle$ with the set of edges $\mathcal{G} = \mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_3 \cup \mathcal{F}_4$ where

$$\mathcal{F}_1 = \text{set of edges of } \mathbb{A}$$

$$\mathcal{F}_2 = \{ (q_*, a, q) : \exists i \in I \ ((i,a,q) \in \mathcal{F}_1) \}$$

$$\mathcal{F}_3 = \{ (q, a, q_*) : \exists t \in T \ ((q,a,t) \in \mathcal{F}_1) \}$$

$$\mathcal{F}_4 = \{ (q_*, a, q_*) : \exists i \in I, \exists t \in T \ ((i,a,t) \in \mathcal{F}_1) \}$$

The restriction of $\mathbb{B}$ to the set of its states which are both accessible and co-accessible yields the trim part of $\mathbb{B}$, which is the desired automaton $\mathbb{A}^*$.

EXAMPLE 6.1.2

Let us consider the alphabet $\mathcal{A}_0 = \{ 0 \ 1 \}$ and the subset $X = \{ 010, 0110, 01110 \}$. The automaton $\mathbb{A}$ which recognizes X and its corresponding automaton $\mathbb{A}^*$ are shown in Fig 6.1.1 and Fig 6.1.2 respectively.

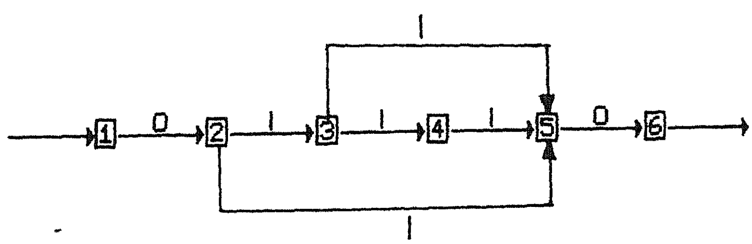


FIGURE 6.1.1. The automaton that recognizes the subset X = {010,0110,01110}

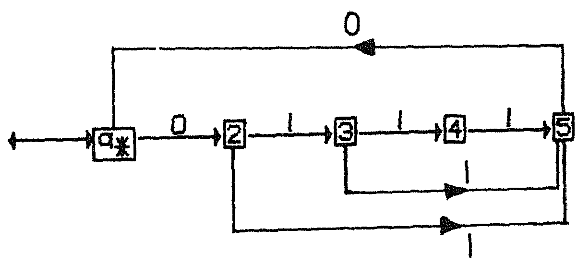


FIGURE 6.1.2. The trim automaton that recognizes X = {010,0110,01110}

EXAMPLE 6 1 3

Let us consider the alphabet $\mathcal{A}_0$ = { 0 1 } and the subset X = 011^*0 The automaton **A** which recognizes X and its corresponding automaton $\mathbf{A}^*$ are shown in Fig 6 1 3 and Fig 6 1 4 respectively

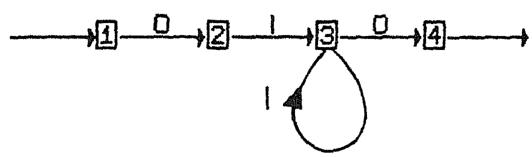


FIGURE 6 1 3 The automaton that recognizes the subset X = 011^*0

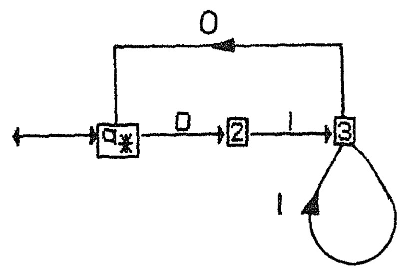


FIGURE 6 1 4 The trim automaton that recognizes the subset X = 011^*0

6 2 AN ALPHABETIC ENCODING OF NORMAL ALGORITHMS

As mentioned earlier, it becomes necessary at times to encode the schemes of certain normal algorithms such that their coded forms become amenable to some other normal algorithms Markov devised a method known as *transcribing a normal algorithm*, by which the scheme of a normal algorithm $\mathcal{N}$ could be coded as a string from a two-lettered alphabet [23]

6 2 1 TRANSCRIPTIONS OF NORMAL ALGORITHMS

In general, the scheme of a normal algorithm $\mathcal{N}$ operating in a certain way on words from a basic alphabet, say, $\mathcal{A}$ is constructed over a larger alphabet $\mathfrak{B}$ where $\mathfrak{B}$ = $\mathcal{A}$ ∪ $\mathcal{E}$ and $\mathcal{E}$ is the alphabet of auxiliary symbols Let $\mathfrak{C}$ be another alphabet where $\mathfrak{C}$ = $\mathfrak{B}$ ∪ { x y z } and the symbols x, y and z are not contained

either in $\mathcal{A}$ or in $\mathcal{B}$ Now the normal algorithm $\mathcal{N}$ is represented in the alphabet $\mathcal{C}$ in the following manner

The substitution formulas of $\mathcal{N}$ are written down in the order of succession of its scheme, placing the symbol z at the end of each substitution formula and replacing all arrows ($\longrightarrow$) and dots ($\cdot$) by the symbols x and y respectively For example, the following scheme

$$\mathcal{N} \qquad \alpha\mu \longrightarrow\cdot \mu\alpha \qquad (\mu \in \mathcal{A} \, , \ \alpha \in \mathcal{B})$$

$$\longrightarrow \alpha$$

could be represented in the alphabet $\mathcal{C}$ as a string of the form :

$$\alpha\mu x y \mu\alpha z x \alpha z$$

Now let us introduce a convenient ordering of symbols in the alphabet $\mathcal{C}$ Let us also denote by r_i , $(1 \leq i \leq n)$ any of the n symbols of $\mathcal{C}$ and translate [Ref Theorem 2 3 2.1] every symbol of $\mathcal{C}$ by using the coding rule Π $r_i \longrightarrow 01^i0$ in the alphabet $\mathcal{A}_0$ = { 0 1 } Π is an injective morphism of the type $\mathcal{C}^* \longrightarrow \mathcal{A}_0^*$ such that $\Pi(\mathcal{C})$ = X is the coding morphism for X = { 01^i0 , $1 \leq i \leq n$ } It is not hard to see that the subset X consisting of (0,1)-links [Definition 2 3.6] is a biprefix code [Definition 6 1 2] over the alphabet $\mathcal{A}_0$ If all the symbols of the string from $\mathcal{C}$ which represents a normal algorithm $\mathcal{N}$ are replaced by their translations by applying the coding rule Π , then the resulting string in $\mathcal{A}_0$ is called the *transcription* of $\mathcal{N}$ and is denoted by $\{\mathcal{N}\}$ For example, let us consider the following scheme

$$\mathcal{N} \cdot \qquad \alpha\mu \longrightarrow\cdot \mu\alpha \qquad (\mu \in \mathcal{A} \, , \ \alpha \in \mathcal{B})$$

$$\longrightarrow \alpha$$

This is represented in the alphabet $\mathcal{C}$ as a string of the form

$$\alpha\mu x y \mu\alpha z x \alpha z$$

Let us now define Π as follows

$$\Pi \qquad x \longrightarrow 010$$

$$z \longrightarrow 0110$$

$$\mu \longrightarrow 01110$$

$$\alpha \longrightarrow 011110$$

$$y \longrightarrow 0111110$$

Now, $(\mathcal{N}) = 0111100111001001111100111001110011100100111100110$

In general, transcription of any normal algorithm over an arbitrary alphabet is a word from the set $\mathfrak{B} = 011^*0$ The set $\mathfrak{B}$ is a subset of the free monoid $\mathcal{A}_0^*$ and so is a code

In what follows, we study some of the important properties of this code

6 2 2 STRUCTURAL PROPERTIES OF THE BIPREFIX CODE $\mathfrak{B} = 011^*0$

PROPOSITION 6 2 2 1

$\mathfrak{B}$ is a biprefix code

PROOF $\mathfrak{B} \subset \mathcal{A}_0^*$ $\mathfrak{B}\mathcal{A}_0^+ \cap \mathfrak{B} = \phi$ $\mathcal{A}_0^+\mathfrak{B} \cap \mathfrak{B} = \phi$ So, $\mathfrak{B}$ is a biprefix code

PROPOSITION 6 2 2 2

The submonoid $\mathfrak{B}^*$ of $\mathcal{A}_0^*$ generated by $\mathfrak{B} = 011^*0$ is free

PROOF $\mathfrak{B} \subset \mathcal{A}_0^*$. Let Π: $\mathbb{C}^* \longrightarrow \mathcal{A}_0^*$ be a coding morphism for $\mathfrak{B}$ Then Π is injective and a bijection from $\mathbb{C}$ into $\mathfrak{B}$ Π is a bijection from $\mathbb{C}^*$ onto $\Pi(\mathbb{C}^*)$ Hence, $\mathfrak{B}^*$ is free and its minimal set of generators $\mathfrak{B}$ is given by $(\mathfrak{B}^*-\Lambda)-(\mathfrak{B}^*-\Lambda)^2$

DEFINITION 6 2 2 1 [6]

Let M be a monoid and N be a submonoid of M Then, N is called *right unitary* in M if $\forall u,v \in M$ ($u,uv \in N \Rightarrow v \in N$), that is , $(N^+)^{-1}N = N$ For convenience, we shall write N^{-1} instead of $(N^+)^{-1}$

N is left unitary in M if $\forall u,v \in M$ ($u,vu \in N \Rightarrow v \in N$), that is, $N(N^+)^{-1} = N$ The submonoid N is called *biunitary* if it is both left and right unitary. N is said to be stable in M if $\forall u,v,w \in M$ ($u,v,uw,wv \in N \Rightarrow w \in N$).

PROPOSITION 6 2 2 3

The free submonoid $\mathfrak{B}^*$ of $\mathcal{A}_0^*$ generated by the set $\mathfrak{B}$ is biunitary and stable in $\mathcal{A}_0^*$

PROOF Let us assume $u=01^i0$, $v=01^j0$ and $w=01^k0$ such that $u,v,w \in \mathcal{A}_0^*$ Now $u,v,uw,wv \in \mathfrak{S}^*$ because 01^i0, 01^j0, 01^i001^k0, $01^k001^j0 \in \mathfrak{S}^*$ and so $w \in \mathfrak{S}^*$ Hence, $\mathfrak{S}^*$ is stable in $\mathcal{A}_0^*$ Using similar arguments, we can see that $\mathfrak{S}^*$ is biunitary in $\mathcal{A}_0^*$

The relationships between the unitariness and stability properties of $\mathfrak{S}$ is shown in Figure 6 2.2 1

$$\begin{tikzcd}[column sep=small]
& \substack{\text{(stability)} \\ (\mathfrak{S}^*)^{-1}\mathfrak{S} \cap \mathfrak{S}(\mathfrak{S}^*)^{-1} = \mathfrak{S}} & \\
\text{(left unitariness) } \mathfrak{S}(\mathfrak{S}^*)^{-1} = \mathfrak{S} \arrow[ur, Rightarrow] & & (\mathfrak{S}^*)^{-1}\mathfrak{S} = \mathfrak{S} \text{ (right unitariness)} \arrow[ul, Rightarrow] \\
& \substack{\mathfrak{S}(\mathfrak{S}^*)^{-1} = (\mathfrak{S}^*)^{-1}\mathfrak{S} = \mathfrak{S} \\ \text{(biunitariness)}} \arrow[ul, Rightarrow] \arrow[ur, Rightarrow] &
\end{tikzcd}$$

FIGURE 6 2.2 1. Relationships between unitariness and stability properties of the code $\mathfrak{S} = 011^*0$

If $X \subset \mathcal{A}^*$ is a code then every subset of X is itself is a code This brings out the notion of *maximal codes* A code is maximal over an alphabet $\mathcal{A}$ if X is not contained in any other code over $\mathcal{A}$. That is, for $X \subset X'$, if X' is a code then $X=X'$ Maximality is not algorithmically verifiable, but it is decidable for recognizable codes Propositions like " Any code X over $\mathcal{A}$ is contained in some maximal code " are provable only by means of nonconstructive methods, which admit the existence of a least upper bound in the set of codes over $\mathcal{A}$ However, as outlined in [6], the extremal properties of codes could be studied in terms of what is known as a *completeness* by replacing an extremal property by a combinatorial one

DEFINITION 6.2.2 2 [6]

Let P be a subset of the free semigroup $\mathcal{A}^+$ where $\mathcal{A}$ is an alphabet An element m in $\mathcal{A}^+$ is *completable* in P, if there exist u and v in $\mathcal{A}^+$ such that umv $\in$P P is said to be *dense* in $\mathcal{A}^+$ if every element of $\mathcal{A}^+$ is completable in P P is said to be *complete* if P^* is dense Every dense set is also complete. A subset of a monoid which is not dense is called *thin set*. All maximal codes are complete All thin complete codes are maximal

PROPOSITION 6 2 2 4

The biprefix code $\mathfrak{S}$ is thin and not maximal

PROOF

$\mathfrak{S}$ would be thin if there is at least one element in $\mathcal{A}_0^*$ which is incompletable in $\mathfrak{S}$ One such element which is incompletable in $\mathfrak{S}$ is 101 So, $\mathfrak{S}$ is thin Moreover, the element 101 is incompletable in $\mathfrak{S}^*$ also This means $\mathfrak{S}^*$ is not dense. Hence $\mathfrak{S}$ is incomplete and not maximal

PROPOSITION 6 2 2 5

Let X_i and X_j be two thin biprefix codes which are finite subsets of $\mathfrak{S}$. Then X_iX_j is thin

PROOF

Let us consider two words u and v in $\mathcal{A}_0^*$ which are incompletable in X_i and X_j respectively Then the word uv is incompletable in X_iX_j . So X_iX_j is thin

DEFINITION 6 2 2 3 [6]

Given a subset N of a monoid M, the set of those elements of M which are factors of elements in N is defined as F(N) where $F(N)=M^{-1}NM^{-1}=\{m \in M \mid \exists\, u,v \in M \cdot umv \in N\}$ and $\bar{F}(N) = M-F(N)$ N is a proper subset of F(N) A code $X \subset \mathcal{A}^+$ is called *very thin* if there exists a word in X^* which is not a factor of any word in X In other words, X is very thin if and only if $X^* \cap \bar{F}(X) \neq \phi$. For example, let us consider the code $X = \{ 0^n10^n \mid n \geq 0 \}$ over the alphabet $\mathcal{A}_0 = \{ 0\ 1 \}$ This code is

very thin because the element 1^2 which is not a factor of any element of X, is present in $X^* \cap \bar{F}(X)$.

PROPOSITION 6.2.2.6

Let X be a subset of the code $\mathfrak{S} = 011^*0$ Then X is very thin

PROOF

Since no other element in $\mathfrak{S}^*$ excepting the set $\mathfrak{S}$, is a factor of any of the elements of $\mathfrak{S}$, $F(\mathfrak{S}) = \mathfrak{S}$ Then, $\bar{F}(\mathfrak{S}) = \mathfrak{S}^* - \mathfrak{S}$ and $\mathfrak{S}^* \cap \bar{F}(\mathfrak{S}) = \bar{F}(\mathfrak{S})$. There is no element in $\bar{F}(\mathfrak{S})$ which is a factor of any of the elements of $\mathfrak{S}$ Hence, $\mathfrak{S}$ is very thin code and every subset of it is also very thin

DEFINITION 6 2 2 4

A code X is *purely thin* if and only if $X^* \cap \bar{F}(X) = \bar{F}(X)$

PROPOSITION 6 2 2 7

The code $\mathfrak{S}$ is purely thin

PROOF The code $\mathfrak{S}$ satisfies the condition $\mathfrak{S}^* \cap \bar{F}(\mathfrak{S}) = \bar{F}(\mathfrak{S})$ So it is purely thin

6 2 3 LITERAL REPRESENTATION OF $\mathfrak{S} = 011^*0$

Literal representation of codes over an alphabet is a convenient method of representing them graphically in the form of a tree whose selected nodes (nodes which are shaded) correspond to words in the code set

Given a finite alphabet $\mathcal{A}$, the literal representation of the corresponding free monoid is constructed in the following manner·

The alphabet is totally ordered and words of equal length are lexicographically ordered The set of words of a specific length form a level in the tree All the nodes of a level are lexicographically labelled from top to bottom The zeroth level consists of a single node known as the root corresponding to the null string. The first right level consists of n nodes where n is the number of symbols in the alphabet Labelling of these n nodes is done from top to bottom in accordance with the lexicographic order introduced in the alphabet. The second

right level consists of n^2 nodes, each being labelled lexicographically from top to bottom with the corresponding n^2 words of length 2 By repeating this procedure the literal representation of the free monoid could be potentially realized For example, the literal representation of the free monoid $\mathcal{A}_0^*$ of the alphabet $\mathcal{A}_0=\{0\,1\}$ is shown in figure 6 2 3 1

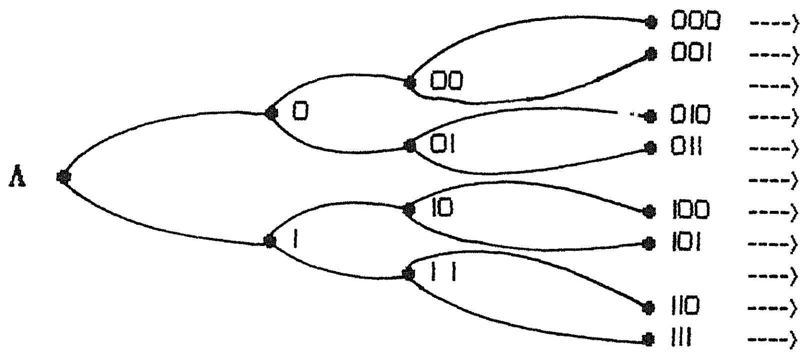


FIGURE 6 2 3 1. Literal representation of the free monoid $\mathcal{A}_0^*$

Since $\mathcal{C}$ is a subset of $\mathcal{A}_0^*$ its literal representation is a subtree of the tree of $\mathcal{A}_0^*$ The subtree corresponding to the code $\mathcal{C}$ is shown in figure 6 2 3 2 [Note The shaded nodes correspond to the words in $\mathcal{C}$]

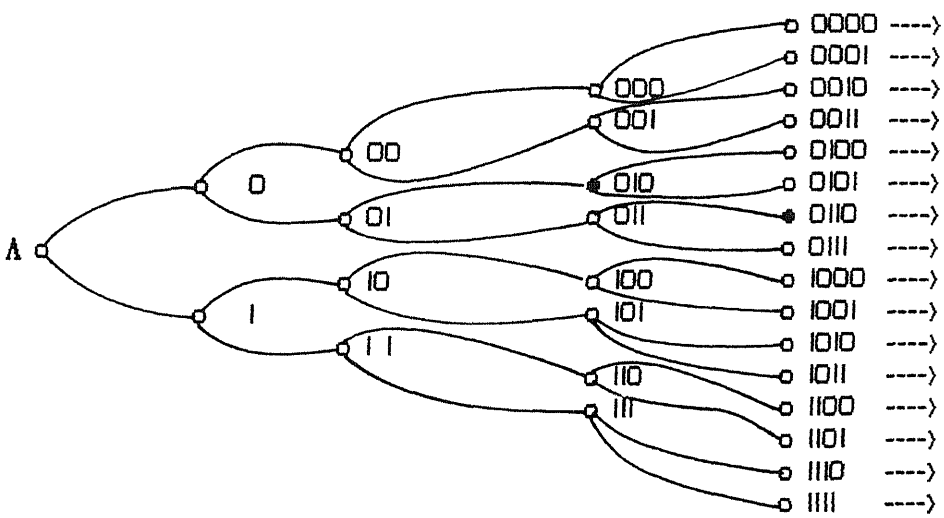


FIGURE 6 2 3 2 Literal representation of the code $\mathcal{C} = 011^*0$

The advantage of the literal representation of codes lies in the easy readability of words and compact diagrammatic representation of reasonably big codes Given a finite alphabet $\mathbb{C}$ [Subsection 6 2 1], the corresponding code set X is also finite Now the literal representation of X yields a method for verifying whether the transcription of a normal algorithm (which is nothing but a (0,1)-chain from the code set $\mathfrak{X}$) is actually in X^* or not In other words, the verification of a valid factor of the transcription of a normal algorithm is done by tracing the path from the root, letter by letter so that a leaf (shaded node) is reached This procedure is repeated for all the factors of the entire coded string The successful completion of all such verifications corresponding to a given (0,1)-chain determines the syntactic correctness of the transcription of a normal algorithm

6 3 THE NOTION OF A CYCLIC NORMAL AUTOMATON

The notion of an unambiguous automaton plays an important role in the study of variable length codes, in the sense that, computations in the monoids of functions are replaced by computations in the monoids of unambiguous relations

DEFINITION 6 3 1 [6]

UNAMBIGUOUS RELATIONS

Let us consider an N-relation [Ref Definition 6 1 5] between two sets P and Q which is a mapping $m : P \times Q \longrightarrow N$ where N is the complete semiring of natural numbers which admits least upper bound The N-relation of m is said to be *unambiguous* if $m \in \{0,1\}^{P \times Q}$ where $0,1 \in N$ Assume P=Q Then a monoid M of unambiguous relations over Q is defined as a submonoid of $\{0,1\}^{Q \times Q}$ with the identity element I_Q The following property holds for M

$$\forall\ m,n \in M \quad p,q \in Q \quad \exists\ r \in Q\ ((p,mn,q)=1 \Rightarrow (p,m,r)=(r,n,q)=1)$$

M is transitive if it satisfies the following property

$$\forall\ p,q \in Q \quad \exists\ m \in M \quad ((p,m,q) = 0 \vee 1)$$

Now, a monoid M of unambiguous partial order relations over a set Q is defined as a submonoid of $\{0,01\}^{Q \times Q}$ with the identity I_Q where 0 and 01 are the constructive natural numbers [Ref Subsection 5 1] corresponding to 0 and 1 from N The system of constructive natural numbers is denoted by the symbol Ч A Ч-relation m is unambiguous only when $m \in \{0,01\}^{Q \times Q}$.

Let us consider a normal algorithm $\mathcal{N}$ over an alphabet $\mathcal{A}$, whose scheme $\mathfrak{S}$ consists of n substitution formulas As described in subsection 2.2., $\mathcal{N}$ is an Ȝ-class semi-Thue presentaion of the free monoid $\mathcal{A}^*$ defined by a totally ordered list of n Ȝ-class partial order relations corresponding to the n substitution formulas Now, one could construct n unambiguous Ч-relations corresponding to the n Ȝ-class partial order relations of the scheme $\mathfrak{S}$ This indicates the possibility of interpreting normal algorithms in terms of unambiguous automata

DEFINITION 6 3 2 · [6]

UNAMBIGUOUS AUTOMATA

Let $\mathbf{A}$ be an automaton over an alphabet $\mathcal{A}$ Then, $\mathbf{A}$ is unambiguous if and only if its representation $\phi_{\mathbf{A}}(\mathcal{A}^*)$ is unambiguous [Ref Definition 6 1 5]. Moreover, if $\mathbf{A} = \langle Q, \Lambda, \Lambda \rangle$ then $\mathbf{A}$ is trim if and only if $\phi_{\mathbf{A}}(\mathcal{A}^*)$ is transitive

DEFINITION 6 3 3

NORMAL AUTOMATON

This is defined as a quintuple $\langle X, Y, Q, \lambda, \delta \rangle$, where X is the set of admissible inputs, Y is the set of outputs, Q is the set of states corresponding to the number of substitution formulas, λ is the next state function of the type $\lambda \; Q \times X \longrightarrow Q$, and δ is the output function of the type $\delta \; Q \times X \longrightarrow Y$ and whose construction is governed by the following rules ·

For any state $q_i \in Q$ let the input string be w_i and the left part of the corresponding i^{th} substitution formula be u_i Then ,

(i) $\lambda(q_i, w_i) = q_0$ if u_i is a factor of w_i , that is , $u_i \leq w_i$

(q_0 is the initial state)

$= q_{i+1}$ otherwise

(ii) $\delta(q_i, w_i) = w_0$ if $u_i \leq w_i$

(w_0 is the input string at the initial state)

$= w_{i+1}$ otherwise

Definition 6 3 3 has been coined with the purpose of interpreting the scheme $\mathfrak{S}$ of a normal algorithm $\mathcal{N}$ over an alphabet $\mathcal{A}$ But, this definition does not fully characterize the class of normal algorithms over an alphabet $\mathcal{A}$, based on their common operational semantics However, the following modified definition gives rise to a syntactic construction which describes the operational scheme of normal algorithms over an alphabet $\mathcal{A}$

DEFINITION 6 3 4

CYCLIC NORMAL AUTOMATON

This is defined as a 7-tuple $\langle X, Y, Q, q_{ini}, q_{fin}, \lambda, \delta \rangle$ where X is the set of admissible inputs, Y is the set of outputs; Q is the set of n states corresponding to the n substitution formulas of the scheme $\mathfrak{S}$ of a normal algorithm $\mathcal{N}$ over an alphabet $\mathcal{A}$, $q_{ini} \notin Q$ is the input state, $q_{fin} \notin Q$ is the output state, λ is the next state function of the type λ $Q \times X \longrightarrow Q$ and δ is the output function of the type δ. $Q \times X \longrightarrow Y$ and whose construction is governed by the following rules For any state q_i , let the input string during the m^{th} iteration of the scheme be w_{mi} and the left part of the corresponding i^{th} substitution formula be u_i

Let us assume that the i^{th} substitution formula is of simple type Then ,

(i) $\lambda(q_i, w_{mi}) = q_0$ if ($u_i \leq w_{mi}$) or $\neg$ ($u_n \leq w_{mn}$)

(q_0 is the first state)

$= q_{i+1}$ otherwise

(ii) $\delta(q_i, w_{mi}) = w_{(m+1)0}$ if ($u_i \leq w_{mi}$) or $\neg$ ($u_n \leq w_{mn}$)

$= w_{m(i+1)}$ otherwise

Let us assume that the i^{th} substitution formula is of terminal type Then ,

(i) $\lambda(q_i, w_{mi}) = q_{fin}$ if $(u_i \leq w_{mi})$

$= q_{i+1}$ otherwise

(ii) $\delta(q_i, w_{mi}) = y \quad y \in Y$ if $(u_i \leq w_{mi})$

$= w_{m(i+1)}$ otherwise

As an illustration, a 6-state cyclic normal automaton corresponding to a scheme consisting of six simple substitution formulas is shown in figure 6 3 1

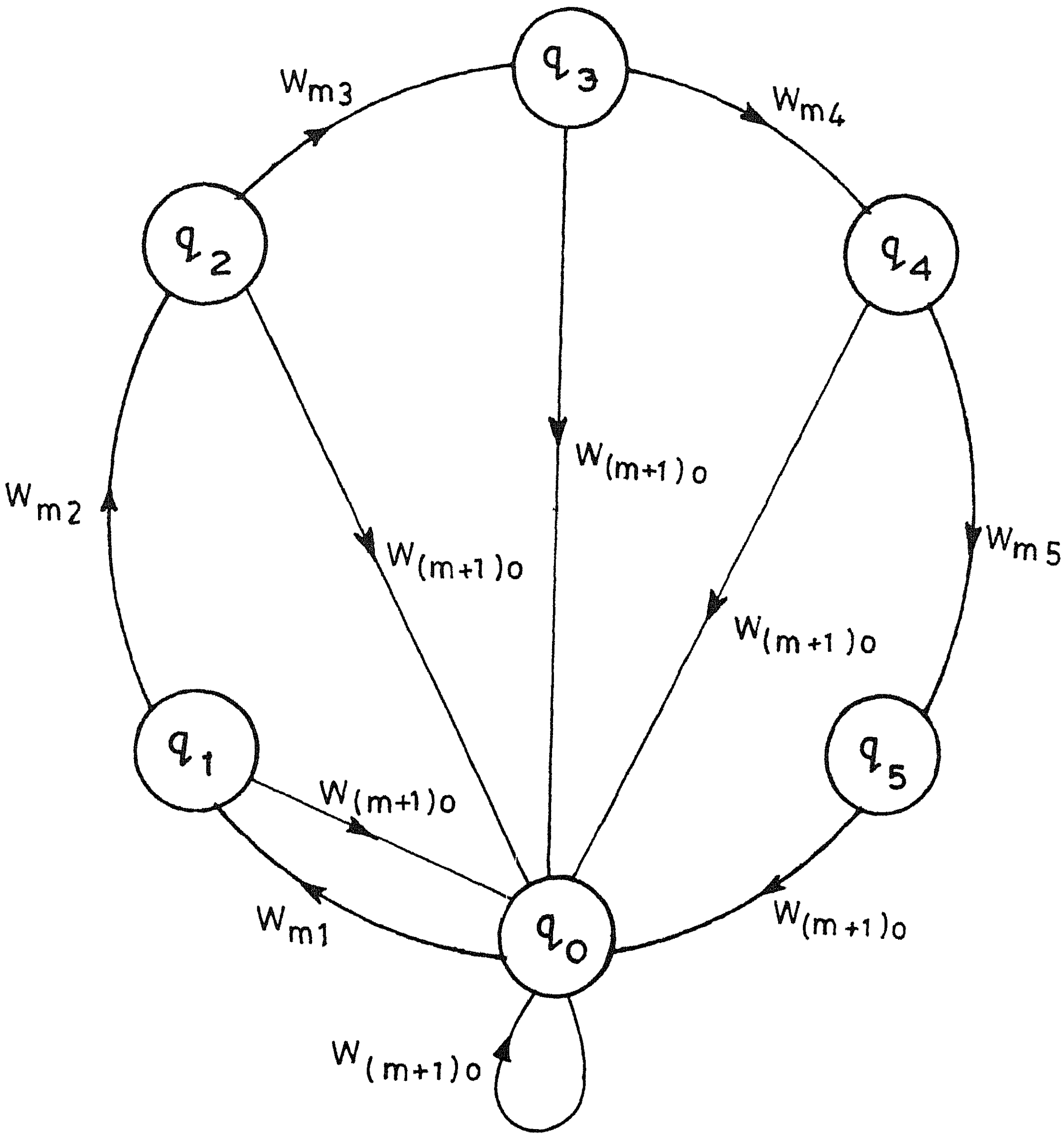


FIGURE 6 3 1 A 6-state cyclic normal automaton

We shall represent a cyclic normal automaton by the symbol $\mathfrak{C}$ Then the representation associated with $\mathfrak{C}$ is a morphism $\phi_{\mathfrak{C}}$ $\mathcal{A}^{*} \longrightarrow \{0,0I\}^{Q \times Q}$, 0,0I ∈ Y such that the following hold

(i) $\phi(\Lambda) = I_Q$ (identity element)

(ii) $(q_i, \phi(w_{mj}), q_j) = 0I$ if there is an edge connecting q_i and q_j.

$= 0$ otherwise

Any cyclic normal automaton $\mathfrak{C}$ over an alphabet $\mathcal{A}$ is unambiguous because, for every $q_i, q_j \in Q$ and $w_{mj} \in \mathcal{A}^{*}$, $(q_i, \phi_{\mathfrak{C}}(w_{mj}), q_j) \in \{0,0I\}$

6 4 IMPLEMENTATION OF SIGNAL PROCESSING OPERATIONS IN TERMS OF REWRITING CYCLIC NORMAL AUTOMATA

In subsection 5 3, it was shown that signal processing operations could be carried out by means of EPT rewriting systems Here, we show that such EPT rewriting systems can be modelled as finite state machines which we shall call as *Rewriting Cyclic Normal Automata* (RCNA) An RCNA is basically a cyclic normal automaton with certain additional factors

DEFINITION 6 4 1

REWRITING CYCLIC NORMAL AUTOMATA

Let us consider a normal algorithm $\mathcal{N}$ consisting of n totally ordered substitution formulas $\{ u_i \longrightarrow v_i \quad 1 \leq i \leq n \}$ and its EPT version We shall define a rewriting cyclic normal automaton for the EPT system in the following manner.

Let Q be the set of states corresponding to the semi-Thue productions contained in the scheme of the EPT rewriting system where $Q = \{q_{ij} \mid 0 \leq i \leq n-1 , 0 \leq j \leq k$ · i represents the i^{th} substitution formula , j represents the j^{th} semi-Thue production of the i^{th} substitution formula and k represents the number of states, that is, the number of semi-Thue productions corresponding to the i^{th} substitution formula$\}$ For convenience, the state corresponding to the i^{th}

substitutiton formula is termed as a *major state* and the states corresponding to the k semi-Thue productions of the i^{th} substitution formula as *minor states* Let $u_i \longrightarrow v_i$ be the i^{th} substitution formula Then, the number of minor states corresponding to the i^{th} major state is given by the following rules

(i) if $|u_i| = |v_i| = \ell_m$, then $k = 2\ell_m$ (NOTE $|u_i|$ is the length of u_i)

(ii) if $|u_i| < |v_i|$ then $k = 2\ell_m$

(iii) if $|u_i| > |v_i|$ then $k = 2\ell_m - 1$

Now, the rewriting cyclic normal automaton is defined as the triple $< Q, INI, FIN >$ where $INI \notin Q$ is the initial state and $FIN \notin Q$ is the final state and whose set of edges $\mathfrak{F}$ contains edges of the following types

(i) $q_{ij} \xrightarrow{\Sigma \backslash \mu / \Leftarrow} q_{(i+1)0}$ (being in the state q_{ij} if the symbol μ is not found in the corresponding input string, go to the next state $q_{(i+1)0}$ and read the same string)

(ii) $q_{ij} \xrightarrow{\mu / \xi} q_{i(j+1)}$ (being in the state q_{ij} if the symbol μ is found in the corresponding input string, recognize it as ξ and go to state $q_{i(j+1)}$)

(iii) $q_{ij} \xrightarrow{\mu / \rightarrow} q_{i(j+1)}$ (being in the state q_{ij} if the symbol μ is found in the corresponding input string, go to state $q_{i(j+1)}$ and read the symbol next to μ)

(iv) $q_{ij} \xrightarrow{\Lambda / \vdash} q_{00}$ (being in the state q_{ij} rewrite the input string with the recognized symbols duely substituted and go to state q_{00})

We shall demonstrate the working of an RCNA by means of the following example

EXAMPLE 6 4 1

Let us consider the EPT rewriting system corresponding to the cyclic shifting normal algorithm $\mathfrak{N}^{cs}$ [Ref Example 5 3 2 2]

The RCNA for the above EPT system is defined by $\mathfrak{C}^{cs} = \langle Q, INI, FIN \rangle$ where

$$
\begin{aligned}
Q \quad = \quad \{ \; & q_{00}, q_{01}, q_{02}, q_{03}, q_{04}, q_{05}, q_{06}, q_{07}, q_{08}, \\
& q_{10}, q_{11}, q_{12}, q_{13}, q_{14}, q_{15}, q_{16}, \\
& q_{20}, q_{21}, q_{22}, q_{23}, q_{24}, q_{25}, q_{26}, \\
& q_{30}, q_{31}, q_{32}, q_{33}, \\
& q_{40}, q_{41}, q_{42}, q_{43}, \\
& q_{50}, q_{51}, q_{52}, q_{53}, \\
& q_{60}, q_{61}, q_{62}, q_{63}, q_{64}, \\
& q_{70}, q_{71}, q_{72}, q_{73}, \\
& q_{80}, \\
& q_{90}, q_{91}, q_{92} \; \}
\end{aligned}
$$

with the set of edges $\mathfrak{F}$ where,

$$
\begin{aligned}
\mathfrak{F} = \{ \; & (q_{INI}, \Sigma^{*}, q_{00}), && (q_{00}, \Sigma\backslash\xi/\Leftarrow, q_{10}), && (q_{00}, \xi/\mu, q_{01}), \\
& (q_{01}, \mu/\rightarrow, q_{02}), && (q_{02}, \Sigma\backslash\alpha/\Leftarrow, q_{10}), && (q_{02}, \alpha/\Gamma, q_{03}), \\
& (q_{03}, \Gamma/\rightarrow, q_{04}), && (q_{04}, \Sigma\backslash\mu/\Leftarrow, q_{10}), && (q_{04}, \mu/\xi, q_{05}), \\
& (q_{05}, \mu/\rightarrow, q_{06}), && (q_{06}, \Sigma\backslash\Gamma/\Leftarrow, q_{10}), && (q_{06}, \Gamma/\alpha, q_{07}), \\
& (q_{07}, \alpha/\rightarrow, q_{08}), && (q_{08}, \Lambda/\vdash, q_{00}), && (q_{10}, \Sigma\backslash\alpha/\Leftarrow, q_{20}), \\
& (q_{10}, \alpha/\mu, q_{11}), && (q_{11}, \mu/\rightarrow, q_{12}), && (q_{12}, \Sigma\backslash\mu/\Leftarrow, q_{20}), \\
& (q_{12}, \mu/\Gamma, q_{13}), && (q_{13}, \Gamma/\rightarrow, q_{14}), && (q_{14}, \Sigma\backslash\Gamma/\Leftarrow, q_{20}), \\
& (q_{14}, \Gamma/\alpha, q_{15}), && (q_{15}, \alpha/\rightarrow, q_{16}), && (q_{16}, \Lambda/\vdash, q_{00}), \\
& (q_{20}, \Sigma\backslash\beta/\Leftarrow, q_{30}), && (q_{20}, \beta/\alpha, q_{21}), && (q_{21}, \alpha/\rightarrow, q_{22}), \\
& (q_{22}, \Sigma\backslash\mu/\Leftarrow, q_{30}), && (q_{22}, \mu/\mu, q_{23}), && (q_{23}, \mu/\rightarrow, q_{24}), \\
& (q_{24}, \Lambda/\beta, q_{25}), && (q_{25}, \beta/\rightarrow, q_{26}), && (q_{26}, \Lambda/\vdash, q_{00}), \\
& (q_{30}, \Sigma\backslash\alpha/\Leftarrow, q_{40}), && (q_{30}, \alpha/\beta, q_{31}), && (q_{31}, \beta/\rightarrow, q_{32}), \\
& (q_{32}, \Sigma\backslash\alpha/\Leftarrow, q_{40}), && (q_{32}, \alpha/\Lambda, q_{33}), && (q_{33}, \Lambda/\vdash, q_{00}), \\
& (q_{40}, \Sigma\backslash\beta/\Leftarrow, q_{50}), && (q_{40}, \beta/\Gamma, q_{41}), && (q_{41}, \Gamma/\rightarrow, q_{42}), \\
& (q_{42}, \Sigma\backslash\beta/\Leftarrow, q_{50}), && (q_{42}, \beta/\Lambda, q_{43}), && (q_{43}, \Lambda/\vdash, q_{00}), \\
& (q_{50}, \Sigma\backslash\Gamma/\Leftarrow, q_{60}), && (q_{50}, \Gamma/\Gamma, q_{51}), && (q_{51}, \Gamma/\rightarrow, q_{52}), \\
& (q_{52}, \Sigma\backslash\Gamma/\Leftarrow, q_{60}), && (q_{52}, \alpha/\Lambda, q_{53}), && (q_{53}, \Lambda/\vdash, q_{00}),
\end{aligned}
$$

$(q_{60}, \Sigma\backslash\Gamma/\Leftarrow, q_{70})$, $(q_{60}, \Gamma/\mu, q_{61})$, $(q_{61}, \mu/\rightarrow, q_{62})$,
$(q_{62}, \Sigma\backslash\mu/\Leftarrow, q_{70})$, $(q_{62}, \mu/\Gamma, q_{63})$, $(q_{63}, \Gamma/\Lambda, q_{64})$,
$(q_{64}, \Lambda/\vdash, q_{00})$, $(q_{70}, \Sigma\backslash\beta/\Leftarrow, q_{80})$, $(q_{70}, \beta/\beta, q_{71})$,
$(q_{71}, \beta/\rightarrow, q_{72})$, $(q_{72}, \Sigma\backslash\alpha/\Leftarrow, q_{80})$, $(q_{72}, \alpha/\Lambda, q_{73})$,
$(q_{73}, \Lambda/\vdash, q_{00})$, $(q_{80}, \Sigma\backslash\Gamma/\Leftarrow, q_{90})$, $(q_{80}, \Gamma/\Lambda, q_{FIN})$,
$(q_{90}, \Lambda/\alpha, q_{91})$, $(q_{91}, \alpha/\rightarrow, q_{92})$, $(q_{92}, \Lambda/\vdash, q_{00})$ }

The graphical representation of $\mathfrak{C}^{cs}$ is shown in Figure 6 4 1

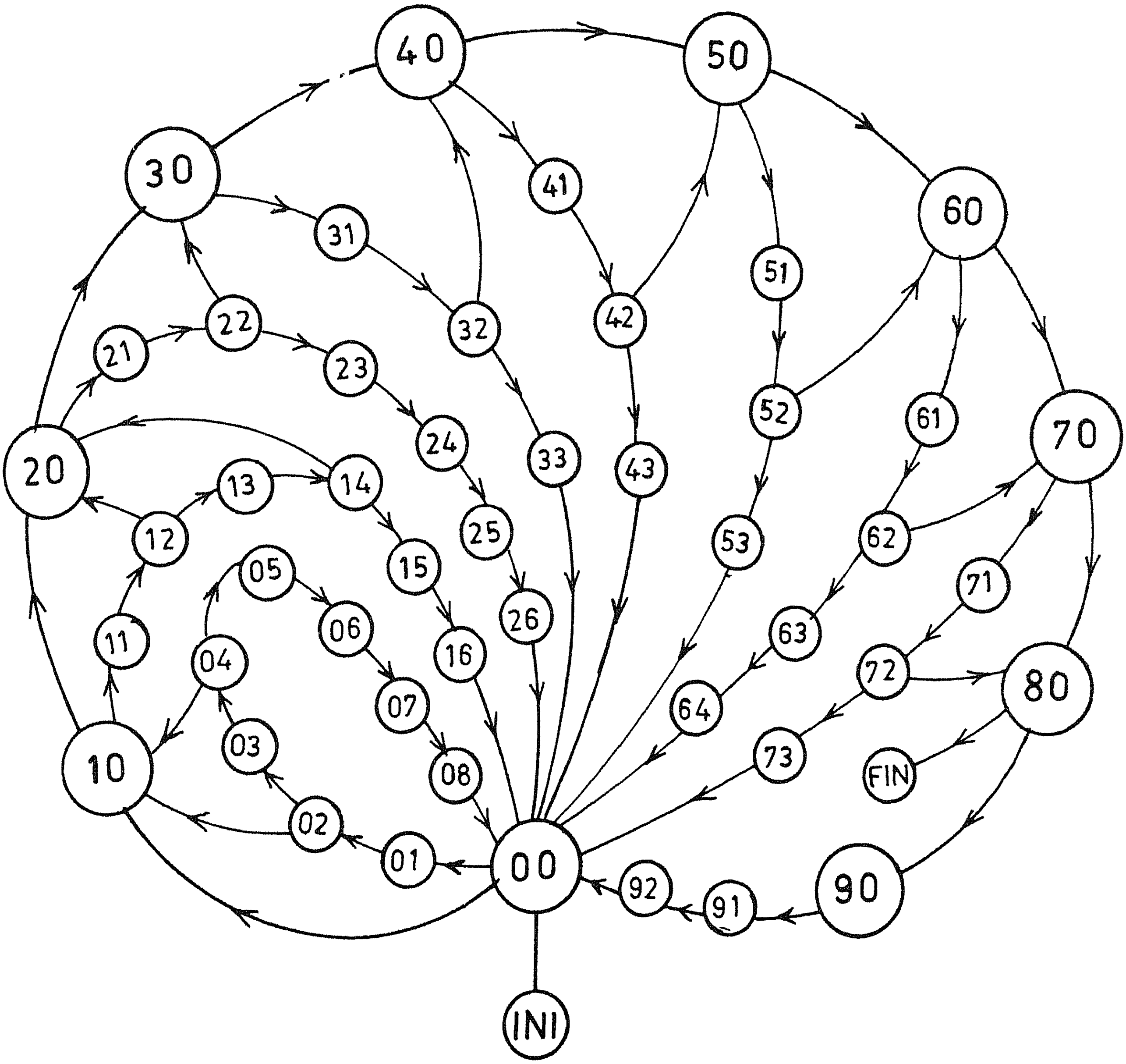


FIGURE 6 4 1. Graphical representation of the rewriting cyclic normal auotmaton

PROPOSITION 6 4 1

The family of languages recognized by Turing machines is also recognized by Rewriting Cyclic Normal Automata

PROOF This is a direct consequence of theorem 5 3 2 1

6.4 1 PRINCIPLE OF NORMALIZATION OF AUTOMATA

With a view to answering the question, in what measure does the precise notion of a normal algorithm relate to the less precise but more general notion of an algorithm in some alphabet, Markov proposed the *principle of normalization* which states

Every algorithm in an alphabet $\mathcal{A}$ is fully equivalent relative to some normal algorithm over $\mathcal{A}$ [23]

This principle asserts that given a well defined algorithm in an alphabet $\mathcal{A}$ with a specific set of input output pairs, one can construct correspondingly an equivalent normal algorithm relative to that alphabet with the same set of input output pairs

In what follows, we interpret this principle in terms of *normal auotmata* [Ref. Definitions 6 3 3 and 6 3 4] and a more general notion of a *family of automata* over an alphabet

In [70], Schutzenberger defines the notion of a family of automata over an alphabet in the following manner.

DEFINITION 6 4 1 1

A family of automata $\mathfrak{A}$ over an alphabet $\mathcal{A}$ consists of machines characterized by the following restrictions

(i) Their output consists in the acceptance (or rejection) of input words belonging to the set F of all words in the letters of a finite alphabet $\mathcal{A}$

(ii) The auotmaton operates sequentially on the successive letters of the input word without the possibility of coming back on the previously read letters

and, thus, all the information to be used in the further computations has to be stored in the internal memory

(iii) The unbounded part of the memory, V_N , is the finite dimensional vector space of the vectors with N integral coordinates, this part of the memory plays only a passive role and all the control of the automaton is performed by the finite part

(iv) Only elementary arithmetic operations are used and the amount of computation allowed for each input letter is bounded in terms of the total number of additions and subtractions.

(v) The rule by which it is decided to accept or reject a given input word is submitted to the same type of requirements and it involves only the storage of a finite amount of information

It is in this context, we propose the following *principle of normalization of automata* which relates the notion of normal automata with that of a family of automata

Every automaton $\mathbf{A}$ belonging to the family of automata $\mathfrak{U}$ over a finite alphabet $\mathcal{A}$ is fully equivalent relative to $\mathcal{A}$ to some normal automaton over $\mathcal{A}$.

This principle highlights the following conjecture

Every family of automata over a potentially realizable alphabet is normalizable and therefore it is impossibile to construct a nonnormalizable automaton

6 4 2 CONSTRUCTION OF FLOWER AUTOMATA FOR TRANSCRIPTIONS OF NORMAL ALGORITHMS

Berstel and Perrin have described in [6], the construction of a universal automaton known as *Flower Automaton* which recognizes a submonoid of $\mathcal{A}^*$ where $\mathcal{A}$ is an alphabet The importance of this type of automaton lies in the fact that it allows a new state $q_* = (\Lambda,\Lambda)$ in its construction by virtue of which one could

describe any type of finite state machine as a flower automaton. The null string Λ is the identity element of the submonoid which is recognized by it

In this subsection, we show that transcripted versions of normal algorithms could be represented by such flower automata

As shown in subsection 6 2, the transcription of a normal algorithm $\mathcal{N}$ is a word over the biprefix code set $\mathfrak{B} = 01{|}^*0$ The characteristic series of $\mathfrak{B}$ denoted by $\underline{\mathfrak{B}}$ is given by

$$(\underline{\mathfrak{B}}, x) = \begin{cases} 1 & \text{if } x \in \mathfrak{B} \\ 0 & \text{else} \end{cases}$$

A trim automaton $\mathbf{A}$ [Ref Def 6 1 3] with a unique initial and a unique final states is unambiguous if and only if its behaviour $|\mathbf{A}|$ [Ref Def 6 1 5] is a characteristic series Now let us construct an automaton $\mathbf{A}_D(\mathfrak{B}) = \langle Q, I, T \rangle$ over the alphabet $\mathcal{A}_0 = \{ 0 \; | \}$ where $Q = \{ (u, v) \in \mathcal{A}_0^* \times \mathcal{A}_0^* | \; uv \in \mathfrak{B} \}$, $I = \Lambda \times \mathfrak{B}$, $T = \mathfrak{B} \times \Lambda$ with edges of the type $(u, v) \xrightarrow{a} (u', v')$ such that $ua = u'$ and $v = av'$ and $uav \in \mathfrak{B}$. The behaviour of this automaton $|\mathbf{A}_D(\mathfrak{B})|$ is equal to the characteristic series $\underline{\mathfrak{B}}$ $\mathbf{A}_D(\mathfrak{B})$ is unambiguous and it recognizes $\mathfrak{B}$ Now we define the flower automaton for the transcription of a normal algorithm in the following manner .

DEFINITION 6 4 2.1

The flower automaton for $\mathfrak{B}$ is defined as the trim part of the automaton $\mathbf{B}_D(\mathfrak{B})$ derived from $\mathbf{A}_D(\mathfrak{B})$ using the canonical construction given in definition 6 1 7. The introduction of a new state $q_* = (\Lambda, \Lambda)$ in the construction of $\mathbf{B}_D(\mathfrak{B})$ allows only four types of edges in the flower automaton

(i) $(u, av) \xrightarrow{a} (ua, v)$ for $uav \in \mathfrak{B}$ and $(u, v) \neq (\Lambda, \Lambda)$

(ii) $(\Lambda, \Lambda) \xrightarrow{a} (a, v)$ for $av \in \mathfrak{B}$ and $v \neq \Lambda$

(iii) $(u, a) \xrightarrow{a} (\Lambda, \Lambda)$ for $ua \in \mathfrak{B}$ and $u \neq \Lambda$

(iv) $(\Lambda, \Lambda) \xrightarrow{a} (\Lambda, \Lambda)$ for $a \in \mathfrak{B}$

EXAMPLE 6 4 2 1

The flower automaton corresponding to the transription 0100110011100111110 is shown in Fig 6 4 2 1

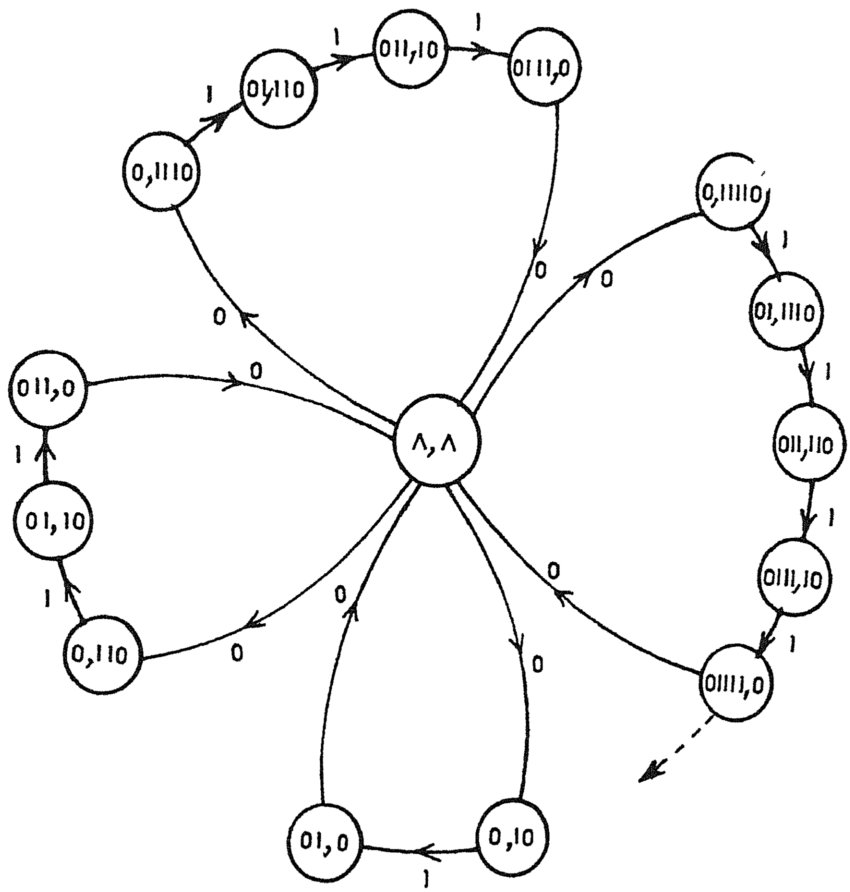


FIGURE 6 4 2 1 The flower automaton corresponding to the transcription 0100110011100111110

PROPOSITION 6 4 2.1

To each cyclic normal automaton we can associate a flower automaton

PROOF

It is a direct consequence of their respective constructions and the transcribability of normal algorithms

PART - III

SECTION 7

THE LOGIC OF CONSTRUCTIVE SIGNAL PROCESSING

The purpose of this section is to study constructive signal processing systems in the framework of a *constructive logic* introduced by Markov [59], [60], [61], [62], [63], [64], [65] This constructive logic is built on a *heirarchical system of languages* denoted by $Я_\alpha$ $\alpha = 0, 1, 2, 3, 4, 5, \ldots, \omega, \omega 1$, whose main constituents are *alphabets, words* and *normal algorithms*

We first present a summary of the basic notions of this logic that are relevant for us and then introduce a theory Th($\mathfrak{R}$) for constructive signal processing In later sections, we make use of this theory in the study of constructive systems in terms of homorphisms and extended topological filters

7 1 THE LANGUAGES $(Я_\alpha)$

7 1 1 LANGUAGE $Я_0$

The starting point of Markov's logic is the language $Я_0$ A *Literoid* is a nonempty word from a one-letter alphabet *Variables* are nonempty words from a one-letter alphabet Literoids and variables are also known as *atoms*. *Verboids* are formed by concatenating literoids to verboids The null string Λ is a verboid By a *term* we mean either a verboid or an atom Let us assume that there occurs a variable X in a term U Substitution of the variable X by another term T is a permissible operation By an *operation*, we mean the action of a normal algorithm $\mathcal{N}$ on a term The result of the operation of substitution of X by T in U by means of a normal algorithm $\mathcal{N}$, is also a term and is denoted by $\mathfrak{F}$[XU□T] where □ is an auxiliary symbol and the symbols [and] play the role of the left and the right

parentheses respectively For example, for any term T, $\mathfrak{F}[XA\Lambda T] \cong \Lambda$ where the symbol $\cong$ stands for *equal by definition*. The notion of *result of the operation of substitution* could be extended to all the remaining languages

An *elementary formula* is a string of the form $T\sigma U$ where T and U are terms and σ is a *comparer* By a *comparer*, we refer to any one of the symbols = or $\neq$ or $\doteq$ or $\not\doteq$ where the symbols = and $\neq$ compare two terms and the symbols $\doteq$ and $\not\doteq$ compare two words or verboids

The only logical connectives that are permitted in $Я_0$ are (i) & (conjunction) and (ii) $\vee$ (disjunction) The quantifier symbols $\forall$ and $\exists$ are allowed in a restricted manner *Quantifiers and connectives are called logical symbols*

Formulas of $Я_0$ are constructed using the following rules

(i) If A is an elementary formula then it is a formula of $Я_0$ We shall designate a formula of $Я_0$ as FmO

(ii) If A and B are FmO's and λ is a connective, then λAB is an FmO

(iii) If A is an FmO, Q is a quantifier, X is a variable, U is a term and β is a limiter (either the symbol $\langle$ or the symbol $\rangle$), then the string $QU\beta XA$ is an FmO.

The variable X is allowed to be substituted only by certain words or verboids For example, the formula $\forall U \langle XA$ where U is a *constant term*, expresses that every formula of the type $\mathfrak{F}_0[XAQ]$ is true where Q is the verboidal prefix of the *meaning of* U Three items are to be explained now (i) Constant Term: It is either the null string Λ or of the form PQ where P is a constant term and Q is a literoid (ii) The meaning of a constant term is always a verboid determined by a normal algorithm (iii) $\mathfrak{F}_0[XAQ]$ is the result of substituting the variable X by a term Q in the formula A which is also a formula To be more specific, the quantification in an FmO is to be understood as that which allows the bounded variable to range in a predetermined, bounded set of words or verboids only

The notion of a *parameter* is important in constructive logic. By parameters

we mean the variables that occur either in an elementary formula or in the formulas A and B contained in the formula λAB or in U and A of the formula QUβXA excluding X The term parameter conveys the same meaning as the term *free variable* conveys in classical logic

As a result, we have the notion of a *closed formula* in constructive logic similar to that of a *sentence* in the classical logic

In general, a closed formula of a language $Я_\alpha$ is a formula without parameters as defined in $Я_\alpha$ Closed formulas of a language $Я_\alpha$ are de ignated as CFα's The set of all formulas of a language $Я_\alpha$ is denoted by {Fmα} and the set of closed formulas by {CFα} {CFα} is a subset of {Fmα} So, {CF0} is the set of all closed formulas in the language $Я_0$ and it is a subset of {Fm0}

The language $Я_0$ does not allow the direct use of a negation symbol in an Fm0 However, it is possible to obtain an Fm0 which is semantically a negative equivalent of another by means of a normal algorithm $\mathcal{N}^{\neg}_{Я_0}$ whose scheme is given below

$\mathcal{N}^{\neg}_{Я_0}$	Substitution Formula		Formula Number
	$\alpha\xi \longrightarrow \xi\alpha$	(ξ denotes every symbol of $Я_0$ other than =, ≠, = and ≠)	(00)
	$\alpha = \longrightarrow = \alpha$		(01)
	$\alpha \neq \longrightarrow \neq \alpha$		(02)
	$\alpha = \longrightarrow = \alpha$		(03)
	$\alpha \neq \longrightarrow \neq \alpha$		(04)
	$\alpha \& \longrightarrow \& \alpha$		(05)
	$\alpha \vee \longrightarrow \vee \alpha$		(06)
	$\alpha \forall \longrightarrow \forall \alpha$		(07)
	$\alpha \exists \longrightarrow \exists \alpha$		(08)
	$\alpha \longrightarrow \cdot$		(09)
	$\longrightarrow \alpha$		(10)

transformed to Q by $\mathcal{N}$

(vi) $(x \Rightarrow \cdot\ y)_{\mathcal{N}}$ is an Fm1 without parameters different from x and y such that $\mathfrak{F}_1[y\mathfrak{F}_1[x(x\Rightarrow\cdot\ y)_{\mathcal{N}}[P]^{\tau}][Q]^{\tau}]$ is a valid CF1 provided P is concludingly transformed to Q by $\mathcal{N}$

(vii) $(x\ y\Rightarrow z)_{\mathcal{N}}$ is a valid Fm1 without parameters different from x, y and z so that $\mathfrak{F}_1[z\mathfrak{F}_1[y\mathfrak{F}_1[x(x\ y\Rightarrow z)(\mathcal{N})][Q]^{\tau}][R]^{\tau}]$ is a valid CF1 provided the verboid $[Q]^{\tau}$ which is the translated version of the word Q from $\mathcal{A}_0 = \{ 0\ |\ \}$, is transformed to the verboid $[R]^{\tau}$ when the transcription of $\mathcal{N}$ is applied to it

An important notion that $\mathcal{A}_1$ introduces is the calculus $\mathfrak{C}$, which is based on the notion of *deducibility* of a CF1 from another. Let S be a finite series of CF1's (that is, a finite number of CF1's are written one after the other in the form of a string) Now we shall call a CF1 C as an immediate consequence of S only in the following cases (i) if there is a valid CF0, (ii) if A and B are in S such that C=&AB, (iii) if either A or B is in S such that C=VAB and (iv) if $\mathfrak{F}_1[XDQ]$ is in S such that C=∃XD Now the notion of deducibility is to be understood in the following manner If S is a deduction and a series of CF1's and C is an immediate consequence from S then SC is a deduction Obviously the null string Λ is a deduction In general, the notion of deducibility is extended to all the remaining languages, in the sense that, for every language $\mathcal{A}_\alpha$ there is a system of rules of deduction S_α using which a CFα could be deduced from another [32]

7 1 3 LANGUAGE $\mathcal{A}_2$

The language $\mathcal{A}_2$ is richer than $\mathcal{A}_1$, in the sense that, it provides rules for the use of *implication* and *negation* of 0^{th} order and *universal quantifier* in constructing Fm2's. Implication is denoted by the symbol ⊃

An Fm2 is constructed using the following rules

(i) If A is an Fm1 then it is also an Fm2

(ii) If A and B are Fm1's then $\supset AB$ is an Fm2

(iii) If A and B are Fm2's and one of them is certainly not an Fm1 then $\&AB$ is an Fm2

(iv) If X is a variable and A is an Fm2 then $\forall XA$ is an Fm2

It is important to note that CF2's cannot be combined using the logical connective of disjunction. Also existential quantifiers cannot be used in the construction of Fm2's

Fm2's of the form $\supset AB$ where A and B are Fm1's are called implications of the 0^{th} order. The implication of the 0^{th} order is to be understood in the following manner. Let S be a series of Fm1's. Let A and B be two Fm1's. Then the Fm2 $\supset AB$ is interpreted as *for an arbitrary S, ((S is a deduction of A) or B)*

The negation of the 0^{th} order is defined as $\neg A \cong A(\neq)$ where A is an Fm1.

The following are the basic deductive rules of the language $\mathcal{A}_2$.

(i) $\dfrac{A \quad \supset AB}{B}$ (ii) $\dfrac{\supset AB \quad \supset BC}{\supset AC}$ (iii) $\dfrac{B}{\supset AB}$ (iv) $\dfrac{\supset AB \quad \supset AC}{\supset A\&BC}$

(v) $\dfrac{\supset AC \quad \supset BC}{\supset VABC}$ (vi) $\dfrac{D \quad E}{\&DE}$ (vii) $\dfrac{\&DE}{D}$ (viii) $\dfrac{\&DE}{E}$

(ix) $\dfrac{\mathfrak{F}_2[XHQ] \text{ for every verboid } Q}{\forall XH}$ (x) $\dfrac{\forall XH}{\mathfrak{F}_2[XHQ]}$ (xi) $\dfrac{\forall X \supset GA}{\supset \exists XGA}$

In addition to these rules, $\mathcal{A}_2$ provides a *semiformal system* S_2 consisting of thirteen rules of deduction which decide the deducibility of a CF2 from another

Let K be a CF2. Let Y be a condition which could be meaningfully imposed on a CF2. Then Y is called *K-inductive* if the following thirteen conditions hold:

(i) K satisfies Y

(ii) Every valid CF2 satisfies Y

(iii) Whenever the CF2's A and ⊃AB satisfy Y, B satisfies Y

(iv) Whenever the CF2's ⊃AB and ⊃BC satisfy Y, then ⊃AC satisfies Y

(v) Whenever the CF2 B satifies Y, then ⊃AB satisfies Y

(vi) Whenever the CF2's ⊃AB and ⊃AC satisfy Y, then ⊃A&BC satisfies Y

(vii) Whenever the CF2's ⊃AC and ⊃BC satisfy Y, then ⊃VABC satisfies Y

(viii) Whenever D and E satisfy Y, then the CF2 &DE satisfies Y

(ix) Whenever the CF2 &DE satisfies Y, then the CF2 . D satisfies Y

(x) Whenever the CF2 &DE satisfies Y, then the CF2 . E satisfies Y

(xi) Whenever we have a general method enabling us to establish for fixed X and H and for any verboid Q that the CF2 $\mathfrak{S}_2$[XHQ] satisfies Y, then the CF2 ∀XH satisfies Y

(xii) Whenever the CF2 ∀XH satisfies Y, then the CF2 $\mathfrak{S}_2$[XHQ] satisfies Y

(xiii) Whenever the CF2 ∀X⊃GA satisfies Y, then the CF2 ⊃∃XGA satisfies Y.

THEOREM 7 1 3 1 [61]

If a condition Y is K-inductive, then every CF2 which is deducible from K satisfies the condition Y

7 1 4 LANGUAGE Я$_3$

This language is just an extension of Я$_2$ in the sense that it provides rules for the use of implication and negation of the first order They are denoted by the symbols ⊃| and ¬| respectively

An Fm3 is constructed using the following rules.

(i) If A is an Fm2 then it is also an Fm3

(ii) If A and B are Fm2's then ⊃|AB is an Fm3

(iii) If C and D are Fm3's such that one of them is certainly not an Fm2, then &AB is an Fm3.

(iv) If X is a variable and E is an Fm3 but not an Fm2 then ∀XE is an Fm3.

The negation of the first order is defined as $\neg|B \cong \supset|B(\neq)$ where B is an Fm2

THEOREM 7 1 4 1 [34]

$\frac{\neg|\neg A}{A}$ where A is an arbitrary Fm1 $\neg A$ is a quasi elementary formula since $\neg A$ is nothing but $\supset A(\neq)$

The above theorem is known as *Markov's Principle of Constructive Choice*

Let K be a CF3 and Y be a condition which could be meaningfully imposed on K Then Y is called 3K-inductive if thirteen conditions similar to the ones given in Я_2 hold N. w, if Y is 3K-inductive, then every CF3 which is 3-deducible [Rt deducibility of CF2's] from K satisfies the condition Y The notion of K-induction could be extended to all the remaining languages in a similar manner.

7 1 5 LANGUAGES Я_4 , Я_5 , .

The languages Я_4 , Я_5 and so on, are the generalizations of the successively extended languages Я_0, Я_1, Я_2 and Я_3 Any language after Я_3 is identified by Я_N N = 4,5,6, with the corresponding system of deductive rules S_N N = 4,5,6,. . For every transition from a language Я_N to its successor Я_{NI} , a new kind of implication emerges, an implication of order N-1 Thus a language Я_N accumulates implications of orders 0 to N-2 However such an abundance of implications does no harm since they agree among themselves

An FmNI is constructed in the strength of the following rules

(i) If A is an FmN then it is also an FmNI

(ii) If A and B are FmN's then $\supset(N-1)AB$ is an FmNI

(iii) If C and D are FmNI's such that one of them is certainly not an FmN, then &CD is an FmNI

(iv) If X is a variable and E is an FmNI but not an FmN then $\forall XE$ is an FmNI

NI-deducibility and NIK-induction are defined in exactly the same manner as they are defined in the previous languages

In general, the negation of order N in the language $Я_{NII}$ is defined as $\neg NA \cong \supset NA(\neq)$ where A is an FmNI

THEOREM 7 1 5 1 [35] $\dfrac{\neg M\neg NA}{A}$ (Proof is ommited here)

7 1 6 LANGUAGE $Я_\omega$

The notion of *abstraction of potential realizability* [23] allows one to unite all the languages so far seen, to form what is known as $Я_\omega$

An Fmω is constructed as per the following rules

(i) If A is an Fm1 then it is also an Fmω

(ii) If C and D are Fmω's and one of them is certainly not an Fm1 then &CD is an FMω

(iii) If E and F are Fmω's then $\supset EF$ is an Fmω

(iv) If X is a variable and E is an Fmω then $\forall XE$ is an Fmω

Fmω's are known as *normal formulas* Two Fmw's cannot be combined disjunctively nor they be existentially quantified However, $Я_\omega$ allows the use of quasi disjunction and quasi existential quantifier in the construction of Fmω's We shall denote quasi disjunction by the symbol $\underline{\vee}$ and the quasi existential quantifier by the symbol $\underline{\exists}$ and agree to their definitions as given in [64], so that the following hold

(i) $\underline{\vee}\, AB \cong \neg\&\neg A\neg B$ where A and B are Fmω's.

(ii) $\underline{\exists}\, XA \cong \neg\forall X\neg A$ where X is a variable and A is an Fmω.

7 1 7 LANGUAGE $Я_{\omega I}$

This language is an extension of $Я_\omega$ and is closed under all traditional logical connectives

FmωI's are constructed by virtue of the following rules

(i) If A is an Fmω then it is also an FmωI

(ii) If A and B are FmωI's but one of them is certainly not an Fmω and λ is a

logical connective either $\supset$ or & then λAB is an Fmωl

(iii) If A is an Fmωl but not an Fmω and X is a variable then $\forall XA$ is an Fmωl

(iv) If A is an Fmωl but not an Fm1 and X is a variable then the string $\exists XA$ is an Fmωl.

The language $\mathcal{A}_1$ is a sublanguage of $\mathcal{A}_{\omega l}$ For any two Fmωl's A and B in which one is certainly not an Fm1 the disjunctive formula $\vee AB$ is defined as $\vee AB \cong \exists z \& \supset (z=) A \supset (z \neq) B$ where z is a variable other than the parameters of A and B

The negation is defined as $\neg A \cong \supset A(\neq)$ where A is an Fmωl.

The rule Modus Ponens holds here *Whenever the CFωl's A and B are such that both A and $\supset AB$ are true in $\mathcal{A}_{\omega l}$ then B is also true in $\mathcal{A}_{\omega l}$*

The principle of constructive choice is stated in $\mathcal{A}_{\omega l}$ as $\supset \forall X \vee D \neg D \supset \neg\neg \exists X D \exists X D$

The above principle plays the fundamental role in the formulation of a constructive theory for signals and systems as we shall see in the next subsection

7.2 NORMAL ALGORITHMIC SIGNAL PROCESSING SYSTEMS REPRESENTED BY A MODEL $C_{\mathfrak{R}}$ OF A CONSTRUCTIVE THEORY $Th(\mathfrak{R})$

Let $\{CF\alpha\}$ be the set of closed formulas of a language $\mathcal{A}_\alpha$. Following standard terminology [3], any subset of $\{CF\alpha\}$ is a *constructive theory*, Φ_α , of that language, and a structure $\mathbb{M}$ is a model of the constructive theory Φ_α if every closed formula of the theory holds in $\mathbb{M}$

Consider now a constructive theory, $Th(\mathfrak{R})$, defined by the following five closed formulas that are $\mathcal{A}_{\omega l}$-provable

$Th(\mathfrak{R})$

(i) Fmωl 1 $\supset \neg\neg ! \mathcal{N}[P] ! \mathcal{N}[P]$

(ii) Fmωl 2 $\supset \mathcal{N}[P] = Q ! \mathcal{N}[P]$

(iii) Fmωl 3 $\supset \forall xyz\, \mathfrak{F}_1 \lfloor z \mathfrak{F}_1 \lfloor y \mathfrak{F}_1 \lfloor x (x\ y \Rightarrow z) \{\mathcal{N}_i\} \rfloor [P]^T \rfloor [Q]^T \rfloor =$

$$\mathfrak{F}_1 \lfloor z \mathfrak{F}_1 \lfloor y \mathfrak{F}_1 \lfloor x (x\ y \Rightarrow z) \{\mathcal{N}_j\} \rfloor [P]^T \rfloor [Q]^T \rfloor \{\mathcal{N}_i\} \simeq \{\mathcal{N}_j\}$$

(iv) Fmωl 4 $\supset \forall XYZ \,\&\&\, \mathcal{N}_i \lfloor X \rfloor = Y \mathcal{N}_j \lfloor Y \rfloor = Z \mathcal{N}_k \lfloor X \rfloor = Z \mathcal{N}_j \lfloor \mathcal{N}_i \lfloor X \rfloor \rfloor = \mathcal{N}_k \lfloor X \rfloor$

(v) Fmωl 5 $\supset \forall XYZ \,\&\&\, \mathcal{N}_i \lfloor X \rfloor = Y \mathcal{N}_j \lfloor X \rfloor = Z \mathcal{N}_k \lfloor X \rfloor = YZ \mathcal{N}_i \lfloor X \rfloor \mathcal{N}_j \lfloor X \rfloor = \mathcal{N}_k \lfloor X \rfloor$

Now, we denote as $C_{\mathfrak{R}}$, the class of signal processing normal algorithms and show that $C_{\mathfrak{R}}$ is a model of $Th(\mathfrak{R})$

Fmωl.1 is a different version of Markov's principle of constructive choice According to this principle, if the assertion of the inapplicability of a normal algorithm $\mathcal{N}$ to some specific word P is refuted then $\mathcal{N}$ is applicable to P By *applicability definiteness* of a normal algorithm $\mathcal{N}$ to a string P, we mean the effective use of atleast one of the substitution formulas of the scheme of $\mathcal{N}$ in rewriting P The *applicability definiteness* (*a-definiteness*) of $\mathcal{N}$ to P is generally expressed as $!\mathcal{N}(P)$ The basic supporting argument underlying Fmωl 1 is the notion of abstraction of potential realizability By virtue of this notion, if the antecedent $\neg\neg !\mathcal{N}\lfloor P \rfloor$ is accepted then the process of applying $\mathcal{N}$ to P cannot continue forever and so it is possible to obtain the result of applying $\mathcal{N}$ to P by actually carrying out the operation of this algorithm step by step waiting for the conclusion of this operation Since computation time and storage space are not considered to be the limiting factors in carrying out normal algorithmic signal processing operations, the formula Fmωl 1 holds for $C_{\mathfrak{R}}$

Fmωl 2 is interpreted in the following manner A normal algorithm $\mathcal{N}$ is said to be a-definite for a word P only when $\mathcal{N}$ is applicable to P and the process of applying it to P terminates naturally or by a terminal substitution formula and the output Q is a word other than P An important question that arises here is of immediate concern to us. Does Q indicate the *desired output* in $\mathcal{N}\lfloor P \rfloor = Q$ of the formula Fmωl.2 ? The answer is negative, due to the fact the a-definiteness of a normal algorithm for a word does not guarantee the transformed word to be the

desired output due to the intended operation for which the very scheme has been constructed For example, let us consider a normal algorithm $\mathcal{N}$ over an alphabet $\mathcal{A}$, whose scheme is constructed with the purpose of carrying out a specific operation on words from $\mathcal{A}$ Let us assume that this scheme contains the simple substitution formula $\longrightarrow \alpha$ Then $\mathcal{N}$ is a-definite for every word in the free monoid $\mathcal{A}^*$ But a-definiteness of $\mathcal{N}$ to the free monoid does not imply that every word of the free monoid is transformed to the relevant output due to the intended operation In order to overcome this difficulty, we shall introduce here the notion of *successful applicability* of a normal algorithm

A normal algorithm $\mathcal{N}$ is said to be *successful applicability definite* (*sa-definite*) for a word P only when the result of applying $\mathcal{N}$ to P is the *desired output* that satisfies the purpose for which the scheme of $\mathcal{N}$ has been constructed

sa-definiteness of a normal algorithm implies its a-definiteness. But the converse is not true always We shall explain this by means of an example. Let us consider the alphabet $\mathcal{A}_2$ = {0 I - /} A word Q from this alphabet is said to represent a valid rational number q if Q is of the form R/S, where, R and S are words that represent integers An integer is a word 0 or any word made up of the only letter I with or without the symbol - left adjoined to it For example, the word -III represents the integer -3 The normal algorithm $\mathcal{N}_Q^{INV}$ whose scheme is given below, is sa-definite for every valid rational number q of the form R/S in the sense that, the result of its application to Q = R/S is the desired output S/R

$\mathcal{N}_Q^{INV}$	substitution formula	formula number
	$0/ \longrightarrow 0$	(00)
	$0I \longrightarrow 0$	(01)
	$/- \longrightarrow -/$	(02)
	$I- \longrightarrow -I$	(03)
	$\alpha- \longrightarrow -\alpha$	(04)
	$\alpha\alpha \longrightarrow \beta$	(05)
	$\beta\alpha \longrightarrow \beta$	(06)
	$\beta/ \longrightarrow /\beta$	(07)
	$\beta 0 \longrightarrow 0\beta$	(08)
	$\beta I \longrightarrow I\beta$	(09)

$$\alpha|| \longrightarrow |\alpha| \qquad (10)$$
$$\alpha|/ \longrightarrow /\alpha| \qquad (11)$$
$$\alpha/| \longrightarrow |\alpha/ \qquad (12)$$
$$\beta \longrightarrow\cdot \qquad (13)$$
$$\longrightarrow \alpha \qquad (14)$$

Let us consider two strings Q = -|||/|||| and Q' = -|||/ and apply $\mathcal{N}_Q^{INV}$ to Q and Q' independently Now, $\mathcal{N}_Q^{INV}$(-|||/||||) = -||||/||| So, $\mathcal{N}_Q^{INV}$ is sa-definite for Q On the other hand, $\mathcal{N}_Q^{INV}$(-|||/) = -/||| This proves that $\mathcal{N}_Q^{INV}$ is certainly a-definite but not sa-definite for Q'

Since all constructive signal processing operations of our interest consist of normal algorithms which are sa-definite for input strings corresponding to admissible signals, proposition 7 2 1 is true for $C_{\mathfrak{R}}$ This means Fmωl 2 is valid for $C_{\mathfrak{R}}$

Fmωl 3 describes the operational equivalence between two normal algorithms $\mathcal{N}_1$ and $\mathcal{N}_j$ over an alphabet $\mathcal{A}$, when the result of the operation of $\mathcal{N}_1$ on a word P is the same as the result of the operation of $\mathcal{N}_j$ on the same word P More precisely Fmωl 3 is read as *if the result of the operation of the transcription of a normal algorithm $\mathcal{N}_1$ in $\mathcal{A}_o^*$ on a word P from an alphabet $\mathcal{A}$, transforming the translated verboid of P in $\mathcal{A}_o^*$ into the translated verboid of Q in $\mathcal{A}_o^*$ is graphically equivalent to the result of the operation of the transcription of a normal algorithm $\mathcal{N}_j$ in $\mathcal{A}_o^*$ on the same word P transforming its translated verboid into the same translated verboid of Q in $\mathcal{A}_o^*$, then the transcripted verboid of $\mathcal{N}_1$ is operationally equivalent to the transcripted verboid of $\mathcal{N}_j$*

Using Fmωl 2 we obtain two formulas from Fmωl.3 : (i) $\supset\mathcal{N}_1[P]=Q!\mathcal{N}_1[P]$ and (ii) $\supset\mathcal{N}_j[P]=Q!\mathcal{N}_j[P]$ Now, one can construct another normal algorithm $\mathcal{N}_k$ over $\mathcal{A}\cup\{*\}$ such that the formula $\mathcal{N}_k[\mathcal{N}_1[P]*\mathcal{N}_j[P]]=\Lambda$ is valid only when $\mathcal{N}_1$ and $\mathcal{N}_j$ are functionally equivalent Fmωl 3 asserts that a particular signal processing operation could be realized by any one of the functionally equivalent constructive systems This allows us to name a constructive signal processing system pertaining

to a particular operation by an abstract label For example, let us agree that the abstract label $\mathfrak{R}^{con}$ refers to a constructive system which implements linear convolution of nonnegative integer sequences This does not mean that $\mathfrak{R}^{con}$ refers only to the scheme given in subsection 4.2 Thus we see that Fmωl 3 is valid for $C_{\mathfrak{R}}$

Fmωl 4 and Fmωl 5 describe the operations of *composition* and *union* of normal algorithms respectively Already in subsection 2 3 3 and in section-4, we have seen how various signal processing operations could be realized by means of constructive systems consisting of normal algorithms combined in an admissible manner by virtue of composition and union theorems So, Fmωl 4 and Fmωl 5 are valid for $C_{\mathfrak{R}}$

Thus, with $C_{\mathfrak{R}}$ as a model of Th($\mathfrak{R}$), we now have a formal basis for the study of the structural properties of $C_{\mathfrak{R}}$-type constructive signal processing systems

SECTION 8

HOMOMORPHISMS IN THE THEORY OF CONSTRUCTIVE SIGNAL PROCESSING

Homomorphisms play an important part in the study of systems One of the important functions of homomorphisms is that they allow us to connect the results of one class of systems to those of an apparently different class of systems. A typical example of such connection to be found in classical systems theory is that provided in Steiglitz [74], where it is shown that the theory of continuous-time and discrete-time systems are connected to each other through a homomorphism induced by the bilinear transformation between the Hilbert spaces $L^2(-\infty, \infty)$ and $\ell_2(-\infty, \infty)$ What such a homomorphism essentially shows is that certain operations, relations and statements relating to these operations and relations are preserved under a particular mapping from $L^2(-\infty, \infty)$ to $\ell_2(-\infty, \infty)$

One important question that arises about homomorphisms in general is: Are *there any general conditions that operations and relations and statements* involving them must satisfy so that they are preserved under homomorphisms ? One way to deal with this question is that of Lyndon [58] who handles it in the framework of logic and algebra

Since we are dealing with the notion of signals and systems in logical terms, Lyndon's results are directly relevant to us In this section, we reformulate Lyndon's homomorphism theorem in terms of constructive mathematical logic and investigate its applications for normal algorithms and constructive signal processing systems

We begin by listing out in the following subsection 8.1, a few relevant properties of various $C_{\mathfrak{R}}$-type constructive signal processing systems Later we refer to this list.

8 1 A SHORT LIST OF CONSTRUCTIVE LOGICAL FORMULAS PERTAINING TO THE MODEL $C_{\mathfrak{R}}$

Table 8 1 1

Sl No	Properties of $C_{\mathfrak{R}}$ relative to an alphabet	constructive logical formula
01	Associativity of words/verboids under *concatenation operation* [Let $\mathcal{N}^{C}$ be the normal algorithm which would concatenate a word to another]	$\forall XYZ\,\mathcal{N}^{C}[\mathcal{N}^{C}[XY]Z]=\mathcal{N}^{C}[X\mathcal{N}^{C}[YZ]]$
02	Concatenation of the identity element (Λ) to a word	$\forall X\,\mathcal{N}^{C}[X\Lambda]=\mathcal{N}^{C}[\Lambda X]=X$
03	Definition of the relation V *is a factor of* X [relation symbol $\preceq$]	$\supset\forall X\exists UVW(X=UVW)V\preceq X$
04	Reflexivity of the relation $\preceq$	$\forall X(X\preceq X)$
05	Antisymmetry of the relation $\preceq$	$\supset\forall XY\&(X\preceq Y)(Y\preceq X)(X=Y)$
06	Transitivity of the realtion $\preceq$	$\supset\forall XYZ\&(X\preceq Y)(Y\preceq Z)(X\preceq Z)$
07	Definition of the relation V *is a proper left factor of* X [relation symbol $<$] [In general, the relation *left factor of* is denoted by the symbol $\leq$]	$\supset\forall X\exists VW\&(X=VW)(W\neq\Lambda)(V<X)$
08	Transitivity of the relation $<$	$\supset\forall XYZ\&(X<Y)(Y<Z)(X<Z)$
09	Definition of the relation W *is a proper right factor of* X [relation symbol. $>$]	$\supset\forall X\exists VW\&(X=VW)(V\neq\Lambda)(W>X)$
10	Transitivity of the relation $>$	$\supset\forall XYZ\&(X)Y)(Y)Z)(X)Z)$

contd..

11	Definition of the relation of *comparability of left factors* [relation symbol $\asymp$]	$\supset\supset\forall X\exists VW\&(V\leq X)(W\leq X)$ $\vee(V\leq W)(W\leq V)(V\asymp W)$
12	Symmetry of the relation $\asymp$	$\supset\forall UV(U\asymp V)(V\asymp W)$
13	Transitivity of the relation $\asymp$	$\supset\forall UVW\&(U\asymp V)(V\asymp W)(U\asymp W)$
14	Definition of the relation of *graphical equivalence* [relation symbol $\doteq$]	$\supset\forall XY\&\supset\forall Z(Z\leq X)(Z\leq Y)$ $\supset\forall Z'(Z'\geq X)(Z'\geq Y)(X\doteq Y)$
15	Reflexivity of the relation $\doteq$	$\forall X(X\doteq X)$
16	Symmetry of the relation $\doteq$	$\supset\forall XY(X\doteq Y)(Y\doteq X)$
17	Transitivity of the relation $\doteq$	$\supset\forall XYZ\&(X\doteq Y)(Y\doteq Z)(X\doteq Z)$
18	Definition of the *equidivisibility of a free monoid'*	$\supset\forall VV'\exists WW'\&(V\asymp V')(VW\doteq V'W')$ $\vee\exists S(V\doteq V'S)\exists T(V'\doteq VT)$
19	*Invariance of words/verboids* by the operation of double inversion [Let $\mathcal{N}^{\sim}$ be the normal algorithm which would invert a word]	$\forall W\,\mathcal{N}^{\sim}[\mathcal{N}^{\sim}[W]]=W$
20	Inversion of concatenation of two words\verboids is the reverse concatenation of their inversions	$\forall VW\,\mathcal{N}^{\sim}[\mathcal{N}^{c}[VW]]\doteq\mathcal{N}^{c}[\mathcal{N}^{\sim}[W]\mathcal{N}^{\sim}[V]]$
21	Inversion of a proper left factor of a word/verboid corresponds to the proper right factor of the inverted word	$\supset\forall VW(V<W)(\mathcal{N}^{\sim}[V]>\mathcal{N}^{\sim}[W])$
22	Definition of the *conjugacy relation between two words* [relation symbol $\triangleleft$]	$\supset\forall XY\&\exists VW(X\doteq VW)(Y\doteq WV)(X\triangleleft Y)$
23	Reflexivity of the relation $\triangleleft$	$\supset\forall X(X\triangleleft X)$

contd..

24	Symmetry of the relation $\lhd$	$\supset\forall XY(X\lhd Y)(Y\lhd X)$
25	Transitivity of the relation $\lhd$	$\supset\forall XYZ\&(X\lhd Y)(Y\lhd Z)(X\lhd Z)$
26	Definition of the relation *elementary transformation of* between two words/verboids by virtue of a normal algorithm $\mathcal{N}$ [relation symbol $\vdash_{\mathcal{N}}$]	$\supset\exists VW\forall XY(\mathcal{N}[XVY]\doteq XWY)(V\vdash_{\mathcal{N}}W)$
27	Reflexivity of the relation $\vdash_{\mathcal{N}}$	$\forall X(X\vdash_{\mathcal{N}}X)$
28	Antisymmetry of the relation $\vdash_{\mathcal{N}}$	$\supset\forall XY\&(X\vdash_{\mathcal{N}}Y)(Y\vdash_{\mathcal{N}}X)(X=Y)$
29	Transitivity of the relation $\vdash_{\mathcal{N}}$	$\supset\forall XYZ\&(X\vdash_{\mathcal{N}}Y)(Y\vdash_{\mathcal{N}}Z)(X\vdash_{\mathcal{N}}Z)$
30	Definition of the relation *transformation of* between words/verboids by virtue of a normal algorithm $\mathcal{N}$. [relation symbol $\models_{\mathcal{N}}$]	$\supset\forall XY\mathcal{N}[X]\doteq Y(X\models_{\mathcal{N}}Y)$
31	Reflexivity of the relation $\models_{\mathcal{N}}$	$\forall X(X\models_{\mathcal{N}}X)$
32	Antisymmetry of the relation $\models_{\mathcal{N}}$	$\supset\forall XY\&(X\models_{\mathcal{N}}Y)(Y\models_{\mathcal{N}}X)(X=Y)$
33	Transitivity of the relation $\models_{\mathcal{N}}$	$\supset\forall XYZ\&(X\models_{\mathcal{N}}Y)(Y\models_{\mathcal{N}}Z)(X\models_{\mathcal{N}}Z)$
34	Definition of the relation *contiguity of* betweeen words/ verboids by virtue of an associative calculus $\mathfrak{A}$. [relation symbol $\perp_{\mathfrak{A}}$	$\supset\exists VW\forall XY\,V(\mathfrak{A}[XVY]\doteq XWY)$ $(\mathfrak{A}[XWY]\doteq XVY)(V\perp_{\mathfrak{A}}W)$
35	Reflexivity of the relation $\perp_{\mathfrak{A}}$	$\forall X(X\perp_{\mathfrak{A}}X)$
36	Symmetry of the relation $\perp_{\mathfrak{A}}$	$\supset\forall XY(X\perp_{\mathfrak{A}}Y)(Y\perp_{\mathfrak{A}}X)$

contd.

37	Transitivity of the relation $\perp_{\mathfrak{R}}$	$\supset\forall XYZ\&(X\perp_{\mathfrak{R}}Y)(Y\perp_{\mathfrak{R}}Z)(X\perp_{\mathfrak{R}}Z)$
38	Definition of the relation *$\mathfrak{R}$-defined equivalence of* between words/verboids by virtue of an associative calculus $\mathfrak{R}$ [relation symbol $\perp\!\!\perp_{\mathfrak{R}}$]	$\supset\forall X\mathfrak{R}[X]=Y(X\perp\!\!\perp_{\mathfrak{R}}Y)$
39	Reflexivity of the relation $\perp\!\!\perp_{\mathfrak{R}}$	$\forall X(X\perp\!\!\perp_{\mathfrak{R}}X)$
40	Symmetry of the relation $\perp\!\!\perp_{\mathfrak{R}}$	$\supset\forall XY(X\perp\!\!\perp_{\mathfrak{R}}Y)(Y\perp\!\!\perp_{\mathfrak{R}}X)$
41	Transitivity of the relation $\perp\!\!\perp_{\mathfrak{R}}$	$\supset\forall XYZ\&(X\perp\!\!\perp_{\mathfrak{R}}Y)(Y\perp\!\!\perp_{\mathfrak{R}}Z)$ $(X\perp\!\!\perp_{\mathfrak{R}}Z)$
42	Universal applicability of the identity normal algorithm $\mathcal{N}^{ID}$ [Ref subsection 2 3 2]	$\forall X\mathcal{N}^{ID}[X]=X$
43	Universal inapplicability of the empty normal algorithm $\mathcal{N}^{NL}$ [Ref subsection 2 3.2]	$\neg!\forall X\mathcal{N}^{NL}[X]=X$
44	Universal applicability of the cyclic shifting normal algorithm $\mathcal{N}^{CS}$ [Ref subsection 4 1 1]	$!\forall X\mathcal{N}^{CS}[X]$

The formulas given in table 8 1 1 are from different languages of $\{\mathcal{A}_\alpha\}$, and they are $\mathcal{A}_{\omega l}$-provable.

8.2 A CONSTRUCTIVE REFORMULATION OF LYNDON'S HOMOMORPHISM THEOREM AND ITS IMPACT ON THE MODEL THEORETIC STUDY OF $C_{\mathfrak{R}}$-SYSTEMS

Lyndon's homomorphism theorem (LHT) asserts that if $\mathfrak{A}$ is a classical algebraic system, P is a first order property true in $\mathfrak{A}$ and ϕ_P is the defining logical sentence of P in the classical first order predicate calculus, then P is

true in all the homomorphic images of $\mathfrak{A}$ if and only if ϕ_p is a positive sentence [46], [58]

Now, the problem of our immediate concern is how to use Lyndon's homomorphism theorem in studying various properties of the model $C_{\mathfrak{N}}$ We shall try to solve this problem by giving a constructive interpretation of Lyndon's homomorphism theorem (LHT), and for its application to various constructive models

To arrive at such an interpretation, we invoke Markov's important result [66] that every $Я_2$-valid closed formula is $Я_\omega$-valid, and that a closed predicate formula (constructive sentence) is deducible in the classical predicate calculus if it is valid either in $Я_2$ or in $Я_\omega$ The converse form of this result is that for every sentence that is constructed in the classical predicate calculus without using the axiom of choice and the law of excluded middle, there is a closed predicate formula which is $Я_2$-valid or $Я_\omega$-valid Thus, if we are able to show that for the sentences that constitute Lyndon's there are corresponding $Я_2$ or $Я_\omega$-valid closed formulas, then we get an interpretation of Lyndon's theorem that we are working for

Now examining the standard statement of Lyndon's theorem, the only term that calls for special attention is the term 'first order property' For languages of $Я_\alpha$ we interpret this term as follows We first, call a closed formula that expresses a property of a constructive system as a *Defining Constructive Sentence* (DCS). A DCS is called positive if all of its predicate symbols and normal algorithmic operations occur positively By a first order DCS, we mean a constructive sentence (closed formula) in which quantification is done only on words and verboids

With these clarifications, we are now in a position to proceed towards interpreting Lyndon's homomorphism theorem in the following manner

8 2 1 FUJIWARA'S VERSION OF LYNDON'S HOMOMORPHISM THEOREM

Lyndon formulated the homomorphism theorem with the help of his *Interpolation Theorem* which is a generalization of Craig's modified version of Gentzen's *Extended Hauptsatz* of Herbrand's theorem [40], [57], [58]

A simpler proof for the homomorphism theorem was given by Keisler in his paper entitled *Theory of models with generalized atomic formulas* [53] The notion of *generalized atomoic (GA) formulas* was introduced by him in order to further generalize model theore ic concepts of subsystems and homomorphisms which are natural generalizations of various notions of modern algebra As a result, more generalized notions such as *F-subsystems* and *F-homomorphisms* appeared when a set of atomic formulas were replaced by a GA-set of formulas The following definitions are useful in order to understand Keisler's version of Lyndon's homomorphism theorem.

DEFINITION 8 2 1.1 [53]

A set F of formulas of a first order language L with identity, is generalized atomic (GA) if the following conditions hold

(i) If $f(x_1, x_2, x_3, \ldots, x_n) \in F$ and $x_1, x_2, x_3, \ldots, x_n$ are mutually distinct, then

$$\left.\begin{array}{l}(1)\ \text{for any variable } y_1 \text{ of } L,\ f(y_1, x_2, x_3, \ldots, x_n) \in F \\ (2)\ \text{for any constant } k \text{ of } L,\ f(k, x_2, x_3, \ldots, x_n) \in F\end{array}\right\}\ \text{closed under substitution}$$

(ii) $f \in F$ and $\vdash f \equiv g$ implies $g \in F$ (closure with respect to logical equivalence.)

(iii) $x_1 = x_2 \in F$ (where x_1 and x_2 are distinct variables)

(iv) The identically false formula $0 \in F$

DEFINITION 8 2.1.2 [53]

Let F be a GA set Then, $\mathfrak{A}$ is said to be an *F-subsystem* of $\mathfrak{B}$ if (i) the carrier of $\mathfrak{A}$ is the subset of the carrier of $\mathfrak{B}$, that is, $A \subseteq B$ and (ii) for any $f \in F$, an

n-tuple $\langle a_1, a_2, a_3, \ldots, a_n \rangle \in A_n$ satisfies f in $\mathfrak{A}$ if and only if it satifies f in $\mathfrak{B}$

DEFINITION 8.2.1.3 [53]

A map h from B onto A is said to be an *F-homomorphism* from $\mathfrak{B}$ to $\mathfrak{A}$ if whenever $f \in F$ and an n-tuple $\langle b_1, b_2, b_3, \ldots, b_n \rangle \in B_n$ satisfies f in $\mathfrak{B}$, $\langle hb_1, hb_2, hb_3, \ldots, hb_n \rangle$ satisfies f in $\mathfrak{A}$. Then $\mathfrak{A}$ is called an F-homomorphic image of $\mathfrak{B}$

DEFINITION 8.2.1.4 [13]

The triple $\langle R, F, \mu \rangle$ is said to be a *signature of an algebraic system if the following conditions hold (i) $R \cap F = \phi$ and (ii) μ : $R \cup F \longrightarrow$* N (set of natural numbers), where R is the set of relations or predicate symbols, F is the set of operation or function symbols, μ is the place or arity mapping

DEFINITION 8.2.1.5 [13]

An algebraic system $\mathfrak{A}$ of a signature $\langle R,F,\mu \rangle$ is an ordered pair $\langle A, \imath^{\mu} \rangle$ where A is the carrier and $\imath^{\mu}$ is known as the *interpretation* of the signature in A, which is understood in the following manner $\imath^{\mu}$ is a mapping of the set $R \cup F$ into relations and operations on the set A. Similarly, if $f \in F$ then, $\imath^{\mu}(f)$ is a $\mu(r)$-place relation on A. Usually, $\imath^{\mu}(r)$ and $\imath^{\mu}(f)$ are written as r^{μ} and f^{μ}

DEFINITION 8.2.1.6 [53]

Let us consider two algebraic systems $\mathfrak{A}$ and $\mathfrak{B}$ with their respective carriers A and B. Let μ be any interpretation in $\mathfrak{A}$ and λ be a unique interpretation in $\mathfrak{B}$ such that μ and λ agree on all variables of the language L. Then, $\mathfrak{B}$ is defined as an *elementary extension* of $\mathfrak{A}$ if for all μ and λ and a formula ϕ of L, if ϕ holds in μ then it holds in λ also. Let F be a GA set. Then, given two systems $\mathfrak{A}$ and $\mathfrak{B}$ we shall say that $\mathfrak{B}$ is an F-extension of $\mathfrak{A}$ only when $\mathfrak{A}$ is an F-subsystem of $\mathfrak{B}$

Keisler's generalized form of Lyndon's homomorphism theorem is as follows

THEOREM 8.2.1.3 [53]

Given an L-system $\mathfrak{B}$ and a GA set F. An L-system $\mathfrak{A}$ has an elementary extension which is F-homomorphic to an elementary extension of $\mathfrak{B}$ if and only if

every sentence positive in F which holds in $\mathfrak{B}$ also holds in $\mathfrak{A}$

[NOTE L refers to the first order language of the classical logic]

Tsuyoshi Fujiwara has modified theorem 8.2.1.3 in the following manner

THEOREM 8.2.1.4 [47]

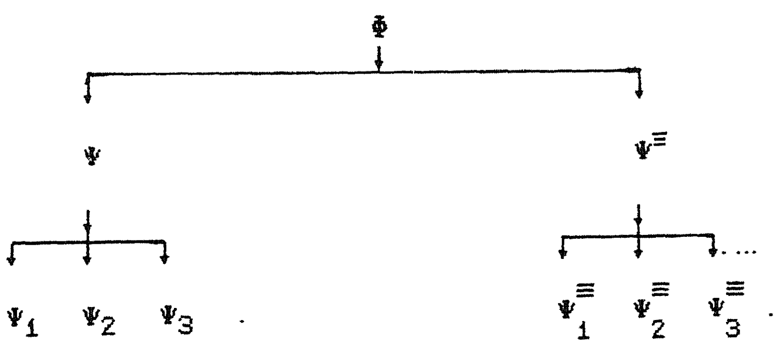
Let L be a first order language with equality. Let F be a GA set of L. Then, $\mathcal{L}F$ refers to the set of all formulas constructed by using the only connectives $\wedge$ and $\vee$ and the quantifiers $\forall$ and $\exists$. Let $\mathfrak{A}$ and $\mathfrak{B}$ be two structures of L. Then, the following two conditions are equivalent (i) Every sentence in $\mathcal{L}F$ that holds in $\mathfrak{A}$ also holds in $\mathfrak{B}$ (ii) There exist an elementary extension of $\mathfrak{A}^*$ of $\mathfrak{A}$ and an elementary extension of $\mathfrak{B}^*$ of $\mathfrak{B}$ such that $\mathfrak{B}^*$ is an F-morphic image of $\mathfrak{A}^*$

8.2.2 INTERPRETATION OF LYNDON'S HOMOMORPHISM THEOREM IN CONSTRUCTIVE ALGEBRAIC LOGIC

In this subsection, we propose a preservation theorem of Lyndon type, based on Fujiwara's version of Lyndon's homomorphism theorem. This preservation theorem is applicable to various $C_{\mathfrak{M}}$-systems.

Firstly, we require the following clarifications

Let Φ be the class of closed predicate formulas that are either $\mathcal{A}_2$ or $\mathcal{A}_\omega$-provable. Then, Φ contains the following



where,

Ψ is the list of constructive sentences without identity $\equiv$

$\Psi^{\equiv}$ is the list of constructive sentences with identity $\equiv$

Ψ_1 is the list of first order constructive sentences without identity in which quantification is done only on elements of sets

$\Psi_1^{\equiv}$ is the list of first order constructive sentences with identity in which quantification is done only on elements of sets.

$\Psi_2^{\equiv}$ and Ψ_2 are lists of second order constructive sentences according as whether the identity is considered or not.

For convenience let us consider an j^{th} order constructive sentence without identity as CS_j and that with identity as CS_{j1} For example, CS_{21} refers to a sentence from the list $\Psi_2^{\equiv}$ Let $\Phi_C = \Psi_1 \cup \Psi_1^{\equiv}$ Let us consider two sets of sentences from Φ_C and denote them by Δ_1 and Δ_2 Let $\mathfrak{R}_1$ and $\mathfrak{R}_2$ with $|\mathfrak{R}_1|$ and $|\mathfrak{R}_2|$ as their carriers, be two systems defined by Δ_1 and Δ_2 respectively Let F be the GE set of CS_1's and CS_{11}'s of $\Delta_1 \cap \Delta_2$ Let M be a subset of $|\mathfrak{R}_1| X |\mathfrak{R}_2|$ Then M is an F-morphism of $|\mathfrak{R}_1|$ onto $|\mathfrak{R}_2|$ if M satisfies the following conditions (i) For any element a in $|\mathfrak{R}_1|$ there is an element b in $|\mathfrak{R}_2|$ such that $\langle a, b\rangle \in M$ (ii) For any element b in $|\mathfrak{R}_2|$ there is an element a in $|\mathfrak{R}_1|$ such that $\langle a, b\rangle \in M$. (iii) For any ϕ_i in F and any element $\langle a_1, b_1\rangle, \langle a_2, b_2\rangle, \ldots, \langle a_n, b_n\rangle$ in M, $\mathfrak{R}_1 \models \phi_i(a_1, a_2, \ldots, a_n)$ implies $\mathfrak{R}_2 \models \phi_i(b_1, b_2, \ldots, b_n)$ An F-morphism is called an F-homomorphism if $\langle a,b\rangle, \langle a,c\rangle \in M$ implies $b = c$ If we consider the constructive sentences without negation from $\Delta_1 \cap \Delta_2$ which form a GE set denoted by $\mathcal{L}F$, then, M is an F-morphism of $|\mathfrak{R}_1|$ onto $|\mathfrak{R}_2|$ implies M is a $\mathcal{L}F$-morphism of $|\mathfrak{R}_1|$ onto $|\mathfrak{R}_2|$

Using arguments analogous to those of Keisler [53], and invoking Markov's result stated earlier [66], we may now conclude that:

THEOREM 8 2 2.1

Let us consider two signal processing systems $\mathfrak{R}_1$ and $\mathfrak{R}_2$ belonging to the model $C_{\mathfrak{R}}$, and let Δ_1 and Δ_2 denote the two sets of closed predicate formulas corresponding to first order properties of the two systems respectively Further let $\mathcal{L}F$ denote the set of closed predicate formulas of $\Delta_1 \cap \Delta_2$ that do not contain negation Then, every sentence in $\mathcal{L}F$ that holds for $\mathfrak{R}_1$ also holds for $\mathfrak{R}_2$

8 2 3 THE PROBLEMS FACED DURING THE HOMOMORPHIC STUDY OF $C_{\mathfrak{R}}$

We faced the following problems while testing theorem 8.2 2 1 by actually applying it to various $C_{\mathfrak{R}}$-systems

(i) Let P be a property of a system $\mathfrak{R}_1$ and ϕ_P be its defining constructive sentence (DCS). If the DCS ϕ_P happens to be negative, then by virtue of the theorem 8.2 2 1 , it is not preserved in any homomorphic image We note that *implication* is a defined concept which makes use of negation and disjunction Does it mean then all the implied statements given in the table 8 1.1 are not preserved under homomorphisms ? Moreover, given a closed predicate formula, is it possible to obtain a logically equivalent GE formula from $\mathcal{L}F$?

(ii) The contrapositive form of Lyndon's theorem is difficult to be proved [46] The converse is interpreted as that a first order property is reflected back from a homomorphic image to the real system if and only if its defining constructive sentence is necessarily a negative sentence Does it mean then, all those properties of the real system which are preserved in a homomorphic image should be definable by a negative constructive sentence in the latter? If it is so, how can we represent, for example, *associativity property* of a system which is preserved in a homomorphic image by a negative sentence ?

(iii) Higher order properties of a constructive system cannot be tested for their validity in a homomorphic image by using theorem 8 2 2 1. For instance, the

notions of *sets* and *numbers* are treated as normal algorithms and any quantification done over them in a closed formula, will lead to a higher order constructive sentence from $\Psi_i \cup \Psi_i^{\equiv}$ It is well known in classical algebraic logic that certain logical tools such as *ultra filters* and *ultra products* could be used in obtaining first order equivalents of certain higher order sentences Is it possible to formulate and use similar constructive logical notions in obtaining first order equivalents of higher order closed formulas ?

In the next section, we shall see that we can as such get rid of these problems by actually resorting to techniques established in Hammer's extended topology in the study of both first order and higher order system properties

SECTION 9

A CONSTRUCTIVE REFORMULATION OF EXTENDED TOPOLOGICAL FILTERS

At the end of section 8, it was mentioned that notions like *ultra filters* and *ultra products* could be used in describing a higher order system property as a first order logical sentence In the classical sense, ultra filters constitute a *special class of topological filters over infinite spaces* These concepts of general topology are not of appropriate use in the case of $C_{\mathfrak{R}}$, because of the fact that they are primarily meant for infinite spaces, whereas the basic spaces of our concern in the study of the model $C_{\mathfrak{R}}$ are finite In view of this fact, it is *more appropriate in our case to use concepts from* Hammer's *extended topology*, because the theory of his extended topology is applicable both to infinite and finite spaces [49], [50]

In this section, we *reformulate the theory of extended topological filters* in constructive logic and show that *a normal algorithm over an alphabet $\mathcal{A}$ is an ordered sequence of constructive extended filter bases over the same alphabet $\mathcal{A}$.*

9 1 THE NOTION OF A CONSTRUCTIVE SET

Topological notions are based on the classical concept of a set, which is commonly understood as a finite or infinite collection of uniquely and simultaneously existing mathematical objects But the concept of a set is not unique in constructive mathematics, for, it depends on the language $Я_{\alpha}$, $0 \leq \alpha \leq \omega !$, chosen to interpret the concept and hence it varies from case to case. We now present the interpretation that is to be used here

In loose terms, a constructive set consists only of constructive objects A

mathematical object of analysis is known as a *constructive object* when it is represented as a word from some alphabet The terms *set* and *property* are considered here to be synonymous A constructive object is said to be a member of a set if it possess the corresponding property

DEFINITION 9 1 1 [20]

Let X_0 denote a collection of some words from an alphabet $\mathcal{A}$ Let $\mathcal{P}_0$ be some property satisfied by the elements of X_0. Then the property $\mathcal{P}_0$ is said to be *potentially enumerable* on X_0 if one could specify another property $\mathcal{P}_1$ which holds for those and only those elements of X_0

Given an alphabet $\mathcal{A}$, if a variable s ranges over the entire free monoid $\mathcal{A}^*$ then s is known as a *base variable* On the other hand, *restricted variables* are defined as those variables whose admissible values (words) are from a finite collection of words from $\mathcal{A}$ Now let us consider a one-parameter formula ϕ_α whose parameter [Ref subsection 7 1] is an individual base variable Hereafter we shall not symbolically indicate the language in which the formulas are written So, ϕ_α is simply written as ϕ ϕ is called an *algorithmically verifiable condition (recursive condition)*, if one can construct some normal algorithm which would verify the fulfillment of the condition imposed by ϕ Let ϕ be a formula of the language $\mathcal{A}_{\omega|}$ of the type $\forall x \mathfrak{B}(x)$ where $\mathfrak{B}(x)$ is a normal formula and x is a restricted variable whose admissible values are from some finite collection, say, X of words from a specific alphabet $\mathcal{A}$ Then one can construct a normal algorithm $\mathcal{N}^{\epsilon}$ which transforms every word chosen by x from X into 0 if $\mathfrak{B}(x)$ holds or into 0| if $\mathfrak{B}(x)$ does not hold Then X is said to be decided by the algorithm $\mathcal{N}^{\epsilon}$ In other words, given an alphabet $\mathcal{A}$ and a property $\mathcal{P}$, one can construct a normal algorithm $\mathcal{N}^{\epsilon}$ over $\mathcal{A}$ which would decide the set X of those constructive objects from $\mathcal{A}$ satisfying the property $\mathcal{P}$ Let us denote for convenience, this normal algorithm by $\mathcal{N}^{\epsilon}_{X}$ Borrowing the terminology from Bishop [7], we shall call this normal

algorithm $\mathcal{N}_X^{\in}$ as the *preset deciding process*

The following example illustrates how one could actually construct the preset deciding process over a specific alphabet for a given set property

EXAMPLE 9 1.1

Alphabet	$\mathcal{A}_0 = \{ 0 \; 1 \}$
Preset to be decided	$\mathbb{N} = \{ 0, 01, 011, 0111, 01111, \ldots \}$
Preset property	$\mathcal{P}$ $\forall x \& (0 \leq x) \neg (10 \leq x)$

$\mathcal{P}$ is a property by which for any word x from $\mathcal{A}_0$ the symbol 0 is a left factor and the string 10 is not a factor

Preset deciding process $\mathcal{N}_{\mathbb{N}}^{\in}$.
$$\begin{cases} 01 \longrightarrow 0 \\ 0 \longrightarrow\cdot \end{cases}$$

[Note This scheme transforms a word from $\mathcal{A}_0$ into a null string Λ if it satisfies $\mathcal{P}$]

A preset might contain more than one identical constructive objects which satisfy the stipulated property whereas such a repetition is not allowed in the notion of a set In order to overcome this difficulty, one can construct a normal algorithm $\mathcal{N}_X^{=}$ which would *regulate a preset* X to contain only unique and nonrepetitive constructive objects, by operating on every pair of elements of X, say, y and z represented as $y*z$ in the following manner

(i) if $y=z$, then $\mathcal{N}_X^{=}(y*z)=y$ and

(ii) if $y \neq z$, then $\mathcal{N}_X^{=}$ is not sa-definite [Ref subsection 7 2] for $y*z$ and the pair $\langle y, z \rangle$ remains unaltered

DEFINITION 9 1 2

A normal algorithm $\mathcal{N}_X^{=}$ is defined as a *regulator of uniqueness in itself of a preset deciding process* $\mathcal{N}_X^{\in}$ if for every y and z from X the following holds

$$\boxed{\forall yz \& \supset (y=z) \mathcal{N}_X^{=} \lfloor y*z \rfloor = y \supset (y \neq z) \neg ! \mathcal{N}_X^{=} \lfloor y*z \rfloor}$$

Based on certain ideas given in [21], we provide the following technique using which one can construct a normal algorithm of the type $\mathfrak{N}^{=}_{X}$

Let $\mathcal{A}$ be an alphabet, ξ and μ be the individual generic variables, P and Q denote two words from $\mathcal{A}$, α and β be the auxiliary symbols and $*$ be the delimiter symbol Let us denote the graphical inverse of a word P as P^{-1} Then one can construct an inverting normal algorithm $\mathfrak{N}^{\sim}$ whose scheme would be of the following type

$\mathfrak{N}^{\sim}$

$$
\begin{array}{ll}
\alpha\xi\mu \longrightarrow \mu\alpha\xi & (\xi, \mu \in \mathcal{A}) \\
\alpha\alpha \longrightarrow \beta & \\
\beta\alpha \longrightarrow \beta & \\
\beta\mu \longrightarrow \mu\beta & \\
\beta \longrightarrow\cdot & \\
\longrightarrow \alpha &
\end{array}
$$

Now let us consider P$*$Q the $*$-system of words from $\mathcal{A}$ One can construct severing normal algorithms $\mathfrak{N}^{*]}$ and $\mathfrak{N}^{[*}$ which would transform P$*$Q in the following manner (i) $\mathfrak{N}^{*]}(P*Q)=Q$ and (ii) $\mathfrak{N}^{[*}(P*Q)=P$ The schemes of both $\mathfrak{N}^{*]}$ and $\mathfrak{N}^{[*}$ are given below

$\mathfrak{N}^{*]}$

$$
\begin{array}{ll}
\mu* \longrightarrow * & (\mu \in \mathcal{A}) \\
* \longrightarrow\cdot &
\end{array}
$$

$\mathfrak{N}^{[*}$

$$
\begin{array}{ll}
*\mu \longrightarrow * & (\mu \in \mathcal{A}) \\
* \longrightarrow\cdot &
\end{array}
$$

Using the composition theorem, [Ref Theorem 2 3 3 1] one can construct a normal algorithm $\mathfrak{N}^{*]\sim}$ such that for any $*$-system P$*$Q of words from $\mathcal{A}$, the

following hold (i) $\mathcal{N}^{*]\sim}(P*Q) \simeq \mathcal{N}^{\sim}(\mathcal{N}^{*]}(P*Q)$ and (ii) $\mathcal{N}^{*]\sim}(P*Q) = Q^{-1}$ Using union theorem, [Ref Theorem 2 3 3 2] one can construct a normal algorithm $\mathcal{N}^{\Gamma}$ such that the following hold for any P*Q

$$(i)\ \mathcal{N}^{\Gamma}(P*Q) \simeq \mathcal{N}^{[*}(P*Q)*\mathcal{N}^{*]\sim}(P*Q) \quad \text{and} \quad (ii)\ \mathcal{N}^{\Gamma}(P*Q) = P*Q^{-1}$$

Now one can construct a normal algorithm $\mathcal{N}^{(=)}$ which would transform a string $P*Q^{-1}$ into an empty string only when P and Q are graphically equivalent Its scheme is given below

$\mathcal{N}^{(=)}$

$$\mu*\mu \longrightarrow * \qquad (\mu \in \mathcal{A})$$

$$* \longrightarrow\!\!\cdot$$

Using the composition theorem, one can construct the desired normal algorithm $\mathcal{N}^{=}$ such that the following hold for any P*Q

(i) $\mathcal{N}^{=}(P*Q) \simeq \mathcal{N}^{(=)}(\mathcal{N}^{\Gamma}(P*Q))$ and

(ii) $\mathcal{N}^{=}(P*Q) = \Lambda$ if P is graphically equivalent to Q

Using the branching theorem, one can construct the normal algorithm $\mathcal{N}^{=}_{X}$ such that the following holds for any P*Q

$$\boxed{\mathcal{N}^{=}_{X}(P*Q) = \mathcal{N}^{[*}(P*Q) = P \ \text{ if } \ \mathcal{N}^{=}(P*Q) = \Lambda}$$

Now, let us consider the following alphabets

(i) $\mathcal{A}_0 = \{\ 0\ |\ \}$ (ii) $\mathcal{A}_1 = \{\ 0\ |\ -\ \}$ (iii) $\mathcal{A}_2 = \{\ 0\ |\ -\ /\ \}$

(iv) $\mathcal{A}_3 = \{\ 0\ |\ -\ /\ \Diamond\ \}$ (v) $\mathfrak{D}_1 = \{\ ,\ \}$ (vi) $\mathfrak{D}_2 = \{\ \Box\ \}$

(vii) $\mathfrak{D}_4 = \{\ \}\ \{\ \}$ (viii) $\mathcal{A}_{11} = \mathcal{A}_0 \cup \mathfrak{D}_1$ (ix) $\mathcal{A}_{15} = \mathcal{A}_{11} \cup \mathfrak{D}_2 \cup \mathfrak{D}_4$

Given a specific property $\mathfrak{P}$ stipulated by means of a formula $\mathfrak{B}(x)$, one can construct the desired set X consisting of objects from a suitable alphabet $\mathcal{A}$ with the help of the normal algorithms $\mathcal{N}^{\in}_{X}$ and $\mathcal{N}^{=}_{X}$ in the following manner

Firstly, one would make a suitable correspondence between the objects of

interest from $\mathcal{A}$ and verboids of the form $0|1^i0$, $i \geq 1$ from $\mathcal{A}_0$ with the help of the transcription theorem [Ref Subsection 6 2 1] By $\{\mathcal{N}_X^{\in}\}$ and $\{\mathcal{N}_X^{=}\}$ we mean the transcriptions of the respective normal algorithms $\mathcal{N}_X^{\in}$ and $\mathcal{N}_X^{=}$

Now, the notion of *constructive set* is defined in the following manner

[NOTE This definition is a variant of Shanin's notion of a constructive set [32]]

DEFINITION 9 1 3

By a constructive set we mean the canonically represented verboid of the form $\{\mathcal{N}_X^{\in}\}\square\{\mathcal{N}_X^{=}\}$ from the alphabet $\mathcal{A}_{15}$ = { 0 | } , { □ } where, $\{\mathcal{N}_X^{\in}\}$ decides the preset by identifying the verboidal transcriptions of those constructive objects which satisfy the property $\mathcal{P}$ stipulated by a one-parameter formula $\mathfrak{B}(x)$ and $\{\mathcal{N}_X^{=}\}$ is the corresponding regulator of uniqueness in itself of $\{\mathcal{N}_X^{\in}\}$

Without loss of generality, we can interpret a set process $\{\mathcal{N}_X^{\in}\}\square\{\mathcal{N}_X^{=}\}$ as $\{\mathfrak{R}_X^{\in}\}\square\{\mathfrak{R}_X^{=}\}$ so as to generalize the notion of a constructive set as the one which is decided by a pair of constructive systems $\mathfrak{R}_X^{\in}$ and $\mathfrak{R}_X^{=}$. The successful termination of the set process $\{\mathfrak{R}_X^{\in}\}\square\{\mathfrak{R}_X^{=}\}$ yields the desired set X which is the string of the form (X†,) where X†, is the ,-dilution of all the unique objects *decided* by the process [Ref Definition 3 1 7]

DEFINITION 9 1 4

A set X is *algorithmically enumerable* if one can construct an algorithm $\mathcal{N}_X^{E}$ over $\mathcal{A} \cup \{0\ |\}$ where $\mathcal{A}$ is an arbitrary alphabet such that for any natural number (a word) n from $\mathcal{A}_0$ and any word P from $\mathcal{A}$, the following hold .

(i) if $!\mathcal{N}_X^{E}(n)$ then $\mathcal{N}_X^{E}(n) \in X$ and

(ii) if $P \in X$ then one can find a natural number i for which $!\mathcal{N}_X^{E}(i)$ and $\mathcal{N}_X^{E}(i) \doteq P$.

Given any arbitrary alphabet, the set of all words of it is enumerable. The following theorem, which we quote without proof, compares the two concepts of algorithmically decidable sets and algorithmically enumerable sets

THEOREM 9 1 1 [20]

Every decidable set is enumerable, but the converse need not be true The intersection of a finite number of enumerable (decidable) sets is also enumerable (decidable) The union of a sequence of enumerable sets is enumerable whereas that of decidable sets can fail to be a decidable set

Based on the definitions 9 1.1 and 9.1 4 , the definition 9.1 3 is restated ~~in the~~ as follows

DEFINITION 9 1 5

By a constructive set, we mean a word of the form $\{\mathfrak{R}_X^{\in}\}\square\{\mathfrak{R}_X^{=}\}$ from the alphabet $\mathcal{A}_{15}$ such that $\{\mathfrak{R}_X^{\in}\}$ decides (enumerates) the verboidal transcriptions of those constructive objects which satisfy the potentially enumerable property $\mathfrak{P}$ stipulated by a formula $\mathfrak{B}(x)$ and $\{\mathfrak{R}_X^{=}\}$ is the corresponding regulator of uniqueness in itself of $\{\mathfrak{R}_X^{\in}\}$

DEFINITION 9 1 6

Let $\mathfrak{P}$ be a potentially enumerable property (PEP) of a set X constructed by $\{\mathfrak{R}_X^{\in}\}\square\{\mathfrak{R}_X^{\in}\}$ Then X is said to be *finite* if the set process $\{\mathfrak{R}_X^{\in}\}\square\{\mathfrak{R}_X^{=}\}$ terminates. On the other hand X is said to be *nonfinite* if the set process is proved to be nonterminating X is said to be *quasifinite*, if one fails to prove that the set process does not terminate

DEFINITION 9 1 7

By the *complement of a set* X of words from an alphabet $\mathcal{A}$ with respect to the entire set of all words from $\mathcal{A}$, we mean the set $\overline{X}$ of words from $\mathcal{A}$ defined by the condition $P\in\overline{X} \equiv \neg(P\in X)$ such that P does not satisfy the corresponding PEP of the set X

DEFINITION 9 1 8

Given two PEP's $\mathfrak{P}_1$ and $\mathfrak{P}_2$ stipulated by formulas $\mathfrak{B}(x)$ and $\mathfrak{B}(y)$ respectively, the constructive sets X and Y are equivalent if and only if all the elements decided by $\{\mathfrak{R}_X^{\in}\}\square\{\mathfrak{R}_X^{=}\}$ satisfy the PEP $\mathfrak{P}_2$ and all the elements decided by $\{\mathfrak{R}_Y^{\in}\}\square\{\mathfrak{R}_Y^{=}\}$ satisfy the PEP $\mathfrak{P}_1$

As outlined by Shanin in [32], various operations on constructive sets and various propositions about them, are the corresponding operations and propositions about formulas

So, the operations of *union* and *intersection* for constructive sets are defined in the following manner

DEFINITION 9 1 9

Given two PEP's $\mathfrak{P}_1$ and $\mathfrak{P}_2$ stipulated by $\mathfrak{B}(x)$ and $\mathfrak{B}(y)$ respectively, the set) is enumerated by $\{\mathfrak{R}_X^{\in}\}\square\{\mathfrak{R}_X^{=}\}$ in the strength of $\mathfrak{P}_1$ and the set Y by $\{\mathfrak{R}_Y^{\in}\}\square\{\mathfrak{R}_Y^{=}$ in the strength of $\mathfrak{P}_2$ such that their union (intersection) could be enumerate by $\{\mathfrak{R}_{X\cup Y}^{\in}\}\square\{\mathfrak{R}_{X\cup Y}^{=}\}$ (and by $\{\mathfrak{R}_{X\cap Y}^{\in}\}\square\{\mathfrak{R}_{X\cap Y}^{=}\}$ respectively) in the strength of th potentially enumerable property $\mathfrak{P}_3$ stipulated by $\mathfrak{B}(z)$ satisfying the followir conditions

(i) $\supset\forall z\mathfrak{B}(z)\bigvee\mathfrak{B}(x)\mathfrak{B}(y)$ (for union operation)

(ii) $\supset\forall z\mathfrak{B}(z)\&\mathfrak{B}(x)\mathfrak{B}(y)$ (for intersection operation)

DEFINITION 9 1 10

Let $\mathfrak{P}_1$ and $\mathfrak{P}_2$ be two PEP's stipulated by $\mathfrak{B}(x)$ and $\mathfrak{B}(y)$ respectively The $\{\mathfrak{R}_X^{\in}\}\square\{\mathfrak{R}_X^{=}\}$ decides (enumerates) the set X in the strength of $\mathfrak{P}_1$ and the set Y decided by the process $\{\mathfrak{R}_Y^{\in}\}\square\{\mathfrak{R}_Y^{=}\}$ in the strength of $\mathfrak{P}_2$. By virtue of definiti 9 1 1, if $\mathfrak{P}_2$ is an enumerable property of words of the basic alphabet $\mathcal{A}$ and if holds for those and only for those words of X which possess the property $\mathfrak{P}_1$, th $\{\mathfrak{R}_Y^{\in}\}\square\{\mathfrak{R}_Y^{\doteq}\}$ decides a particular subset of the set X In such a case, we shall s that the property $\mathfrak{P}_2$ is *imbeddable* in $\mathfrak{P}_1$ Given a PEP $\mathfrak{P}$, for every one of

imbeddable property $\mathfrak{P}_1$ there corresponds a set which is a subset of the set correspnding to $\mathfrak{P}$

As already mentioned, for a given PEP $\mathfrak{P}$ defined by a formula $\mathfrak{B}(x)$, the corresponding set X decided by the process $\{\mathfrak{R}_X^{\in}\}\Box\{\mathfrak{R}_X^{=}\}$ is a word of the form {Xt,} from the alphabet $\mathcal{A}_{15}$= { 0 I } , { □ } One can construct a normal algortihm $\mathcal{N}_X^{\nu}$ which would find out a constructive natural number n corresponding to every word of the form {Xt,} indicating the number of ,-terms in the word, which is in other words known as the cardinal nu.·ber of the set X

DEFINITON 9 1 11

Two constructive sets X and Y corresponding to the PEP's $\mathfrak{P}_1$ and $\mathfrak{P}_2$ defined by $\mathfrak{B}(x)$ and $\mathfrak{B}(y)$ are disjoint if one could specify an enumerable property $\mathfrak{P}_3$ by a formula $\mathfrak{B}(z)$ so that the following condition holds

$$\supset \forall z \mathfrak{B}(z) \neg (\& \mathfrak{B}(x) \mathfrak{B}(y))$$

This means, for any two disjoint constructive sets X and Y the following conditions hold

(i) $$\{\mathfrak{R}_{X\cap Y}^{\in}\}\Box\{\mathfrak{R}_{X\cap Y}^{=}\} = \Lambda$$

(ii) $$\left\{ \mathcal{N}^{\nu}_{\{\mathfrak{R}_{X\cap Y}^{\in}\}\Box\{\mathfrak{R}_{X\cap Y}^{=}\}} \right\} = \left\{ \mathcal{N}^{+}_{\{\mathcal{N}_X^{\nu}\},\{\mathcal{N}_Y^{\nu}\}} \right\}$$

(ii) $$\left\{ \mathcal{N}^{\nu}_{\{\mathfrak{R}_{X\cup Y}^{\in}\}\Box\{\mathfrak{R}_{X\cup Y}^{=}\}} \right\} = \left\{ \mathcal{N}^{+}_{\{\mathcal{N}_X^{\nu}\},\{\mathcal{N}_Y^{\nu}\}} \right\}$$

The algorithm $\mathcal{N}^{+}$ of the type shown below is used to add constructive natural numbers

$\mathcal{N}^{+}$	Substitution formulas	Formula number
	$,0 \rightarrow ,$	(0)
	$, \rightarrow \cdot$	(1)

In a similar manner, one can construct a normal algorithm $\mathcal{N}^{-}$ like the one shown below which would subtract a constructive natural number from another

$\mathcal{N}^{-}$	Substitution formulas	Formula number
	$I,I \rightarrow ,$	(0)
	$I,0 \rightarrow I,$	(1)
	$0,0 \rightarrow 0,$	(2)
	$0,I \rightarrow ,-0I$	(3)
	$, \rightarrow \cdot$	(4)

With the help of the normal algorithms $\mathcal{N}^{\nu}$, $\mathcal{N}^{+}$ and $\mathcal{N}^{-}$ one can construct the following equalities which connect the cardinalities of any two arbitrary sets X and Y

$$\text{(i)} \quad \left\{\mathcal{N}^{\nu}_{\{\mathfrak{R}^{\in}_{X\cap Y}\}\square\{\mathfrak{R}^{=}_{X\cap Y}\}}\right\} = \left\{\mathcal{N}^{-}_{\{\mathcal{N}^{+}_{\{\mathcal{N}^{\nu}_{X}\},\{\mathcal{N}^{\nu}_{Y}\}}\},\{\mathcal{N}^{\nu}_{\{\mathfrak{R}^{\in}_{X\cup Y}\}\square\{\mathfrak{R}^{=}_{X\cup Y}\}}\}}\right\}$$

$$\text{(ii)} \quad \left\{\mathcal{N}^{\nu}_{\{\mathfrak{R}^{\in}_{X\cup Y}\}\square\{\mathfrak{R}^{=}_{X\cup Y}\}}\right\} = \left\{\mathcal{N}^{-}_{\{\mathcal{N}^{+}_{\{\mathcal{N}^{\nu}_{X}\},\{\mathcal{N}^{\nu}_{Y}\}}\},\{\mathcal{N}^{\nu}_{\{\mathfrak{R}^{\in}_{X\cap Y}\}\square\{\mathfrak{R}^{=}_{X\cap Y}\}}\}}\right\}$$

Now, the notion of a power set is interpreted in the following manner: With reference to definition 9 1 10, given a set property $\mathcal{P}_1$ which is potentially

enumerable, one can specify an enumerable property $\mathcal{P}_2$ which is imbeddable in $\mathcal{P}_1$ so that the corresponding set of $\mathcal{P}_2$ becomes the subset of the set corresponding to $\mathcal{P}_1$ In what follows, we suggest a method by which one can construct the power set using the idea of property imbeddability

An enumerable property $\mathcal{P}_2$ is said to be *imbeddable* in a given PEP $\mathcal{P}_1$ if and only if $\mathcal{P}_2$ holds for those and only those elements of the set described by the property $\mathcal{P}_1$ Let us assume that $\mathcal{P}_1$ and $\mathcal{P}_2$ which are defined by one-parameter formulas, say, $\mathfrak{B}_1(x)$ and $\mathfrak{B}_2(x)$ respectively, are restriction-free In other words, these enumerable properties are not stipulated with any restriction in deciding, for example, the number of elements which satisfy them In such a case, the imbedding of $\mathcal{P}_2$ in $\mathcal{P}_1$ is called *free imbedding* On the other hand, if $\mathfrak{B}_2(x)$ is formulated along with its validity subjected to one or more constraints and $\mathcal{P}_2$ is imbeddable in $\mathcal{P}_1$, then the imbedding is called *restricted imbedding* For example, if the restriction is about the cardinality of the set to be decided then the corresponding restricted imbedding is called *cardinality restricted imbedding* Now the *power set of a given constructive set* is obtained as follows.

Let X_0 be the constructive set decided by $\{\mathfrak{R}^{\in}_{X_0}\}\square\{\mathfrak{R}^{=}_{X_0}\}$ in the strength of the PEP $\mathcal{P}_0$ by the formula $\mathfrak{B}$ [Note The restricted variable is not shown here in the logical formula] Now one can specify enumerable properties of the types $\mathcal{P}_{10}$, $\mathcal{P}_{11}$, $\mathcal{P}_{12}$, , $\mathcal{P}_{1n}$ where n is the cardinality of the base set X_0 Thus one can specify ${}^nC_0$ number of properties of the type $\mathcal{P}_{10}$, ${}^nC_1$ number of independent properties of the type $\mathcal{P}_{11}$, ${}^nC_2$ number of independent properties of the type $\mathcal{P}_{12}$ and so on, up to ${}^nC_n$ number of properties of the type $\mathcal{P}_{1n}$ In general, one can specify potentially infinite enumerable properties $\mathcal{P}^k_{ij}$, where i is the i^{th} power of the base set, j is the cardinality index that ranges from 0 to the cardinal number (n_{i-1}) of the $(i-1)^{th}$ power set and k is the index that ranges from 1 to ${}^{(n_{i-1})}C_j$ $j \geq 0$ for every $\mathcal{P}_{ij}$ and for every $i \geq 1$ [Table 9 1 1]

Table 9 1 1 Potentially enumerable properties corresponding to the power set of a constructive set

Sl No	Subset properties	Number of properties	Remarks
1	$\mathcal{P}^{1}_{10}$	1	Any property which holds for none of the elements of the base set X_0
2	$\mathcal{P}^{1}_{11}$, $\mathcal{P}^{2}_{11}$, , $\mathcal{P}^{k}_{11}$, , $\mathcal{P}^{n}_{11}$	n	$\mathcal{P}^{k}_{11}$ implies any abstract $\mathcal{P}_0$ imbeddable property that holds only for the k^{th} element of the base set X_0
3	$\mathcal{P}^{1}_{12}$, $\mathcal{P}^{2}_{12}$, , $\mathcal{P}^{k}_{12}$,	${}^{n}C_2$	$\mathcal{P}^{k}_{12}$ implies any abstract $\mathcal{P}_0$ imbeddable property that holds only for the k^{th} pair of elements of X_0
4	$\mathcal{P}^{1}_{13}$, $\mathcal{P}^{2}_{13}$, , $\mathcal{P}^{k}_{13}$,	${}^{n}C_3$	$\mathcal{P}^{k}_{13}$ implies any abstract $\mathcal{P}_0$- imbeddable property that holds for the k^{th} triple of the set X_0.
	↓	↓	↓
2^n	$\mathcal{P}^{1}_{1n}$	1	Any $\mathcal{P}_0$-imbeddable property that holds to all ν elements of the base set X_0

DEFINITION 9.1.12

Let $\mathcal{P}_0$ be the given PEP in the strength of which $\{\Re^{\in}_{X_0}\}\square\{\Re^{=}_{X_0}\}$ decides the constructive set X_0. Let $\mathcal{P}^k_{ij}$ denote any abstract cardinality restricted, $\mathcal{P}_0$-imbeddable enumerable property. Let $\{\mathcal{N}^{\nu}_{X_0}\} = n$. Then the power set X_1 of the base set X_0 is of the form $(X_1 t,)$ which is decided by the process $\{\Re^{\in}_{X_1}\}\square\{\Re^{=}_{X_1}\}$ using the property $\mathcal{P}_1$ which is a system of 2^n $\mathcal{P}_0$-imbeddable properties

$$\mathcal{P}^k_{1j} \quad 0 \le j \le n \,,\; 1 \le k \le {}^nC_j$$

Based on the above details, we now reformulate the theory of extended topological filters in the constructive logic.

9.2 CONSTRUCTIVE EXTENDED TOPOLOGICAL FILTERS

In 1937, Cartan formulated the concepts of filters and ultra filters which were later developed by Bourbaki, Schmidt and Grimeisen. The development of the theory associated with these concepts took place as a part of the study of general topology. But these mathematical tools of general topology have been found wanting in the study of various problems of topological nature in areas like control systems, signal processing, logic and computing, and system theory. This is due to the fact that the general topology deals only with infinite spaces [44].

In 1961, Hammer brought various extensions in general topology such that his theory could be applied to finite sets also. A filter in general, as defined by Hammer, is an ordered dichotomy over a set <accepted elements, rejected elements> [44]. Let us take for instance a normal algorithm $\mathcal{N}$ over an alphabet $\mathcal{A}$. $\mathcal{N}$ is a filter in the sense that it divides the free monoid $\mathcal{A}^*$ into two disjoint sets $\$^{!\mathcal{N}}$ and $\$^{\neg!\mathcal{N}}$. $\$^{!\mathcal{N}}$ is the set of all strings in $\mathcal{A}^*$ to which $\mathcal{N}$ is sa-definite [Ref subsection 7.2] and $\$^{\neg!\mathcal{N}}$ is its complement with respect to $\mathcal{A}^*$. So the normal algorithmic filter $\mathcal{N}$ is represented by the ordered dichotomy $\langle \$^{!\mathcal{N}}, \$^{\neg!\mathcal{N}} \rangle$.

Following this notion of a general filter, Thampuran developed the notion of an *extended Filter* which is a generalization of the concept of a filter due to Cartan [75] An extended filter over a finite set is a set which consists only of elements from the power set of the given set other than the null set such that every element present in the filter set ensures the presence of every one of its super sets which are elements of the power set

9 2 1 DEFINITION OF A CONSTRUCTIVE EXTENDED FILTER

On the same lines of Thampuran, we think of a constructive extended filter over a base set X_0 as a property $\mathfrak{P}_F$ described by a formula $\mathfrak{B}(f)$ where $\mathfrak{P}_F$ is the system of the union of certain cardinality restricted $\mathfrak{P}_0$-imbeddable subset properties such that every such property present in the system $\mathfrak{P}_F$ ensures the presence of all those cardinality restricted $\mathfrak{P}_0$-imbeddable properties which would form a linear chain with it as the basis, by the partial ordering of imbeddability

A formal definition of *constructive extended filter* is given below

DEFINITION 9 2 1 1

By a *constructive proper extended filter over a base set* X_0, we mean a set F which is decided by the process $\{\mathfrak{R}^{\in}_{F}\}\square\{\bar{\mathfrak{R}}^{=}_{F}\}$ in the strength of the filter property $\mathfrak{P}_F$ depicted by the formula $\mathfrak{B}(f)$ such that the following conditions hold for F

(i) If a set decided by the process $\{\mathfrak{R}^{\in}_{X_{1j},k}\}\square\{\bar{\mathfrak{R}}^{=}_{X_{1j},k}\}$ is a filter element, then every element $\{\mathfrak{R}^{\in}_{X_{1j'},k'}\}\square\{\bar{\mathfrak{R}}^{=}_{X_{1j'},k'}\}$ in the power set $\{\mathfrak{R}^{\in}_{X_1}\}\square\{\bar{\mathfrak{R}}^{=}_{X_1}\}$ where $j\neq j'$ and $k \neq k'$ and for which the element $\{\mathfrak{R}^{\in}_{X_{1j},k}\}\square\{\bar{\mathfrak{R}}^{\doteq}_{X_{1j},k}\}$ is a subset due to the restricted imbedding of the property $\mathfrak{P}^{k}_{1j}$ in the property $\mathfrak{P}^{k'}_{1j'}$ is also a filter element

(ii) $\mathfrak{P}^{1}_{10}$ does not belong to the system of properties $\mathfrak{P}_F$

The definition of *constructive improper filter* would be obtained if the

condition (ii) in the definition 9 2 1 , is replaced by the following one

$$\mathfrak{P}^{i}_{10} \text{ belongs to the system of properties } \mathfrak{P}_F$$

However, we are concerned here only with constructive proper extended filters and hereafter we shall call them simply as *constructive extended filters*

DEFINITION 9 2 1 2

A constructive extended filter (CEF) is called a *constructive cartan filter* if it satisfies an additional condition that the intersection of any two filter elements is also a filter element

9 2 1 1 SPECIFIC RESULTS CONCERNING CONSTRUCTIVE CARTAN FILTERS

Let us consider a base set X_0 over an alphabet $\mathcal{A}$ Let $\left\{\mathfrak{R}^{\in}_{F_1}\right\}\square\left\{\mathfrak{R}^{=}_{F_1}\right\}$ and $\left\{\mathfrak{R}^{\in}_{F_2}\right\}\square\left\{\mathfrak{R}^{=}_{F_2}\right\}$ be two constructive cartan filters defined over X_0 in the strength of the filter properties $\mathfrak{P}_{F_1}$ and $\mathfrak{P}_{F_2}$ Now if $\mathfrak{P}_{F_1}$ is $\mathfrak{P}_{F_2}$-imbeddable, then $\left\{\mathfrak{R}^{\in}_{F_2}\right\}\square\left\{\mathfrak{R}^{=}_{F_2}\right\}$ is said to be *finer than* $\left\{\mathfrak{R}^{\in}_{F_1}\right\}\square\left\{\mathfrak{R}^{=}_{F_1}\right\}$ In other words, $\left\{\mathfrak{R}^{\in}_{F_1}\right\}\square\left\{\mathfrak{R}^{=}_{F_1}\right\}$ is said to be *coarser than* $\left\{\mathfrak{R}^{\in}_{F_2}\right\}\square\left\{\mathfrak{R}^{=}_{F_2}\right\}$

Let us now consider three cartan filters F_1, F_2 and F_3 over X_0 described by the PEP's $\mathfrak{P}_{F_1}$, $\mathfrak{P}_{F_2}$ and $\mathfrak{P}_{F_3}$ respectively Let us also assume that $\mathfrak{P}_{F_1}$ is $\mathfrak{P}_{F_2}$-imbeddable and $\mathfrak{P}_{F_3}$-imbeddable and $\mathfrak{P}_{F_2}$ is $\mathfrak{P}_{F_3}$-imbeddable Then, F_2 is said to be the *interpolant filter* of F_1 and F_2

Given two cartan filters F_i and F_j , the latter for instance is said to be a *just finer filter* of the former only when the following two conditions are satisfied

(i) $\mathfrak{P}_{F_i}$ must be a $\mathfrak{P}_{F_j}$-imbeddable property

(ii) There is no interpolant filter between F_i and F_j.

Let us consider n cartan filters over X_0 where $\left\{\mathfrak{R}^{\in}_{F_i}\right\}\square\left\{\mathfrak{R}^{=}_{F_i}\right\}$ $1\leq i\leq n$ be the i^{th} CEF These n CEF's are said to form a *linear filter chain* if and only if for every i^{th} CEF, the $(i+1)^{th}$ CEF is a just finer filter

restricted $\mathfrak{F}_0$-imbeddable properties such that the power set X_1 is obtained as

$$\left\{\mathfrak{R}^{\in}_{X_1}\right\}\square\left\{\mathfrak{R}^{=}_{X_1}\right\} = \{\{\},\{010\},\{0110\},\{01110\},\{010,0110\},\{010,01110\},\{0110,01110\},\{010,0110,01110\}\}$$

Now, three ultra filters could be obtained by virtue of three independent $\mathfrak{F}_0$-imbeddable ultra filter properties

(i) $\left\{\mathfrak{R}^{\in}_{\tilde{F}_1}\right\}\square\left\{\mathfrak{R}^{=}_{\tilde{F}_1}\right\} = \{\{010\},\{010,0110\},\{010,01110\},\{010,0110,01110\}\}$

(ii) $\left\{\mathfrak{R}^{\in}_{\tilde{F}_2}\right\}\square\left\{\mathfrak{R}^{=}_{\tilde{F}_2}\right\} = \{\{0110\},\{010,0110\},\{0110,01110\},\{010,0110,01110\}\}$

(iii) $\left\{\mathfrak{R}^{\in}_{\tilde{F}_3}\right\}\square\left\{\mathfrak{R}^{=}_{\tilde{F}_3}\right\} = \{\{01110\},\{010,01110\},\{0110,01110\},\{010,0110,01110\}\}$

The family of all constructive cartan filters over X_0 is given in table 9.2.1.1.1 and the corresponding lattice structure is shown in figure 9.2.1.1.1

Table 9.2.1.1.1 Constructive cartan filters over a set $X = \{010,0110,01110\}$

Sl No	constructive cartan filters
1	$\left\{\mathfrak{R}^{\in}_{\tilde{F}_1}\right\}\square\left\{\mathfrak{R}^{=}_{\tilde{F}_1}\right\} = \{\{010\},\{010,0110\},\{010,01110\},\{010,0110,01110\}\}$
2	$\left\{\mathfrak{R}^{\in}_{\tilde{F}_2}\right\}\square\left\{\mathfrak{R}^{=}_{\tilde{F}_2}\right\} = \{\{0110\},\{010,0110\},\{0110,01110\},\{010,0110,01110\}\}$
3	$\left\{\mathfrak{R}^{\in}_{\tilde{F}_3}\right\}\square\left\{\mathfrak{R}^{=}_{\tilde{F}_3}\right\} = \{\{01110\},\{010,01110\},\{0110,01110\},\{010,0110,01110\}\}$
4	$\left\{\mathfrak{R}^{\in}_{F_{11}}\right\}\square\left\{\mathfrak{R}^{=}_{F_{11}}\right\} = \{\{010,0110\},\{010,0110,01110\}\}$
5	$\left\{\mathfrak{R}^{\in}_{F_{21}}\right\}\square\left\{\mathfrak{R}^{=}_{F_{21}}\right\} = \{\{010,01110\},\{010,0110,01110\}\}$
6	$\left\{\mathfrak{R}^{\in}_{F_{31}}\right\}\square\left\{\mathfrak{R}^{=}_{F_{31}}\right\} = \{\{0110,01110\},\{010,0110,01110\}\}$
7	$\left\{\mathfrak{R}^{\in}_{F_c}\right\}\square\left\{\mathfrak{R}^{=}_{F_c}\right\} = \{\{010,0110,01110\}\}$

$$
\begin{tikzcd}[column sep=small]
\{\mathfrak{R}^{\in}_{\tilde{F}_1}\}\square\{\mathfrak{R}^{=}_{\tilde{F}_1}\} \arrow[d, dash] \arrow[dr, dash] & \{\mathfrak{R}^{\in}_{\tilde{F}_2}\}\square\{\mathfrak{R}^{=}_{\tilde{F}_2}\} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \{\mathfrak{R}^{\in}_{\tilde{F}_3}\}\square\{\mathfrak{R}^{=}_{\tilde{F}_3}\} \arrow[dl, dash] \arrow[d, dash] \\
\{\mathfrak{R}^{\in}_{F_{11}}\}\square\{\mathfrak{R}^{=}_{F_{11}}\} \arrow[dr, dash] & \{\mathfrak{R}^{\in}_{F_{21}}\}\square\{\mathfrak{R}^{=}_{F_{21}}\} \arrow[d, dash] & \{\mathfrak{R}^{\in}_{F_{31}}\}\square\{\mathfrak{R}^{=}_{F_{31}}\} \arrow[dl, dash] \\
 & \{\mathfrak{R}^{\in}_{F_C}\}\square\{\mathfrak{R}^{=}_{F_C}\} &
\end{tikzcd}
$$

FIGURE 9 2 1 1 1 Lattice formed by the constructive cartan filters over $X_0 = \{010, 0110, 01110\}$

9.2 1.2 GENERAL RESULTS CONCERNING CONSTRUCTIVE EXTENDED FILTERS

Given a constructive set $\{\mathfrak{R}^{\in}_{X_0}\}\square\{\mathfrak{R}^{=}_{X_0}\}$ with the property $\mathcal{F}_0$, its constructive power set $\{\mathfrak{R}^{\in}_{X_1}\}\square\{\mathfrak{R}^{=}_{X_1}\}$ possesses a relational structure of a complete lattice in which the lower bound of two subsets is their intersection and the upper bound is their union A complete lattice is a partially ordered set in which every nonempty subset has a supremum and an infimum

EXAMPLE 9 2 1 2 1

Let us consider a set of constructive objects from an alphabet $\mathcal{A}$ which is decided by the process $\{\mathfrak{R}^{\in}_{X_0}\}\square\{\mathfrak{R}^{=}_{X_0}\}$ in the strength of the property $\mathcal{F}_0$ such that $\{\mathfrak{R}^{\in}_{X_0}\}\square\{\mathfrak{R}^{=}_{X_0}\} = \{010, 0110, 01110, 011110\}$ Then the power set property $\mathcal{F}_1$ consists of 16 properties of the type $\mathcal{F}^{k}_{1j}$ $0 \le j \le 4$ and $1 \le k \le {}^4C_j$ which are $\mathcal{F}_0$-imbeddable such that 16 constructive subsets of the type $\{\mathfrak{R}^{\in}_{X_{1j}^k}\}\square\{\mathfrak{R}^{=}_{X_{1j}^k}\}$ [Ref. table 9.2 1 2 1] form the power set X_1.

Table 9 2 1 2 1 Subsets of a constructive set

Sl No	Constructive subsets of $\left\{\mathfrak{R}^{\in}_{X_{0}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{0}}\right\}$ = (010,0110,01110,011110)
1	$\left\{\mathfrak{R}^{\in}_{X_{14}^{1}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{14}^{1}}\right\}$ = (010,0110,01110,011110)
2	$\left\{\mathfrak{R}^{\in}_{X_{13}^{1}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{13}^{1}}\right\}$ = (010,0110,01110)
3	$\left\{\mathfrak{R}^{\in}_{X_{13}^{2}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{13}^{2}}\right\}$ = (010,0110,011110)
4	$\left\{\mathfrak{R}^{\in}_{X_{13}^{3}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{13}^{3}}\right\}$ = (010,01110,011110)
5	$\left\{\mathfrak{R}^{\in}_{X_{13}^{4}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{13}^{4}}\right\}$ = (0110,01110,011110)
6	$\left\{\mathfrak{R}^{\in}_{X_{12}^{1}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{12}^{1}}\right\}$ = (010,0110)
7	$\left\{\mathfrak{R}^{\in}_{X_{12}^{2}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{12}^{2}}\right\}$ = (010,01110)
8	$\left\{\mathfrak{R}^{\in}_{X_{12}^{3}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{12}^{3}}\right\}$ = (010,011110)
9	$\left\{\mathfrak{R}^{\in}_{X_{12}^{4}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{12}^{4}}\right\}$ = (0110,01110)
10	$\left\{\mathfrak{R}^{\in}_{X_{12}^{5}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{12}^{5}}\right\}$ = (01110,011110)
11	$\left\{\mathfrak{R}^{\in}_{X_{12}^{6}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{12}^{6}}\right\}$ = (0110,011110)
12	$\left\{\mathfrak{R}^{\in}_{X_{11}^{1}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{11}^{1}}\right\}$ = (010)
13	$\left\{\mathfrak{R}^{\in}_{X_{11}^{2}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{11}^{2}}\right\}$ = (0110)
14	$\left\{\mathfrak{R}^{\in}_{X_{11}^{3}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{11}^{3}}\right\}$ = (01110)
15	$\left\{\mathfrak{R}^{\in}_{X_{11}^{4}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{11}^{4}}\right\}$ = (011110)
16	$\left\{\mathfrak{R}^{\in}_{X_{10}^{1}}\right\}\square\left\{\mathfrak{R}^{=}_{X_{10}^{1}}\right\}$ = ()

The complete lattice formed by the 16 constructive subsets is shown in figure 9 2 1 2 1

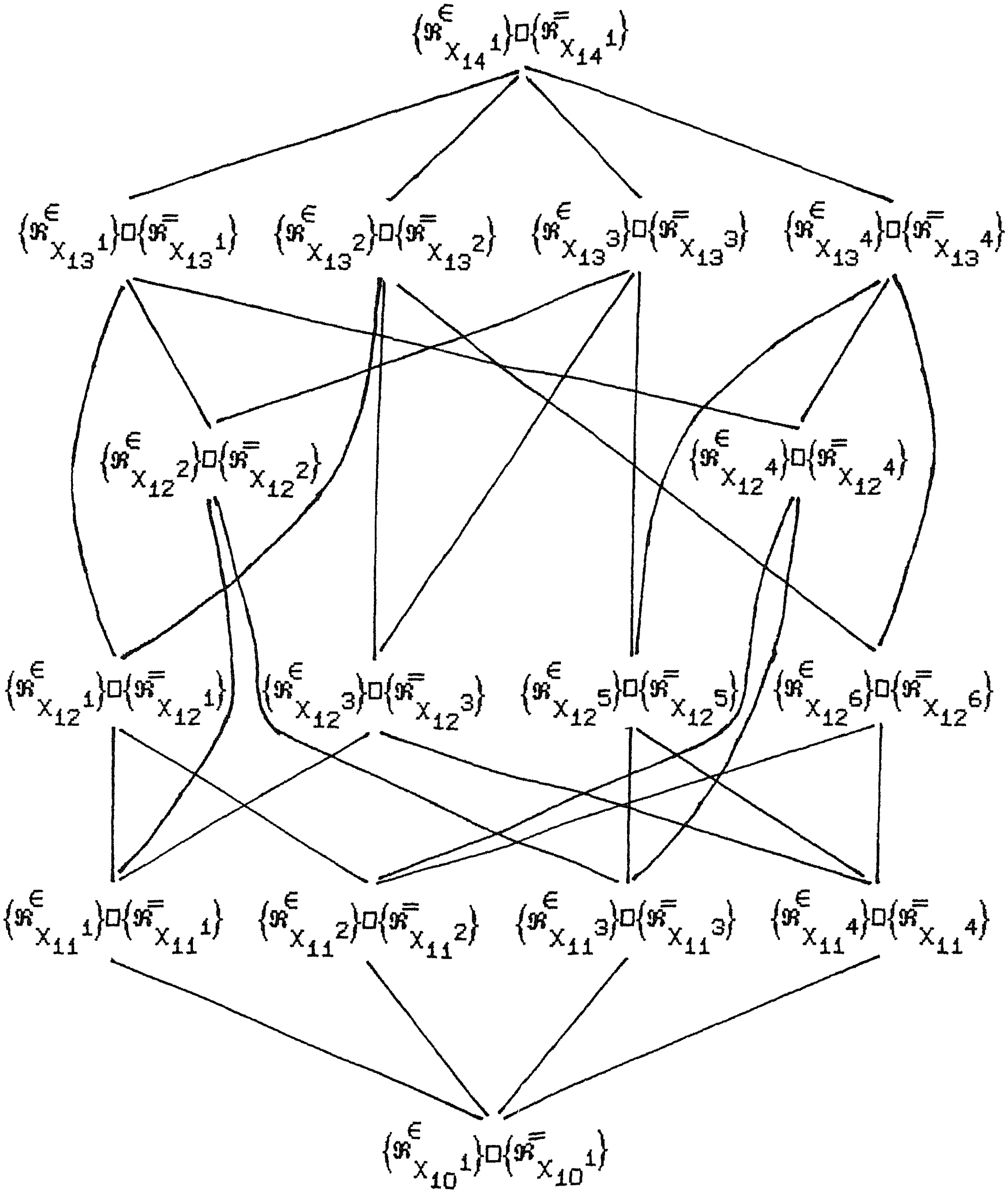


FIGURE 9 2 1 2 1 Lattice formed by the subsets of a constructive set

THEOREM 9.2.1.2.1

Given a constructive base set $\{\Re^{\in}_{X_0}\}\square\{\Re^{=}_{X_0}\}$ with the property $\mathcal{P}_0$, let us denote by Φ, the family of all constructive extended filters over the base set. Then Φ possesses a relational structure of a complete distributive lattice which is denoted by $(\Phi, \subseteq)$ where $\subseteq$ is the relation 'coarser than'

PROOF

The condition for any lattice to be complete is that every one of its elements should have a supremum and an infimum. This means the union and intersection of nonempty elements of the lattice should also be the lattice elements. In the same way we can assert that $(\Phi, \subseteq)$ is a complete lattice, by proving that the union and intersection of any nonempty family of CEF's are also CEF's. Let us consider n CEF's which are certain elements of the lattice under consideration. Then in the strength of theorem 9.1.1 and definition 9.1.9, one can construct the following processes

(i) $$\left\{\Re^{\in}_{\bigcap_{i=1}^{n}\left(\{\Re^{\in}_{F_i}\}\square\{\Re^{=}_{F_i}\}\right)}\right\}\square\left\{\Re^{=}_{\bigcap_{i=1}^{n}\left(\{\Re^{\in}_{F_i}\}\square\{\Re^{=}_{F_i}\}\right)}\right\}$$ and

(ii) $$\left\{\Re^{\in}_{\bigcup_{i=1}^{n}\left(\{\Re^{\in}_{F_i}\}\square\{\Re^{=}_{F_i}\}\right)}\right\}\square\left\{\Re^{=}_{\bigcup_{i=1}^{n}\left(\{\Re^{\in}_{F_i}\}\square\{\Re^{=}_{F_i}\}\right)}\right\}$$

which would enumerate the intersection and union of the n CEF's. The intersection of n CEF's is also a CEF due to the following reason. Every element of the constructive set of the intersection of n CEF's is present in all the n CEF's. This is obvious due to the very definition of intersection. By virtue of the definition

of a proper extended filter, every such element which is present in all the n CEF's ensures the presence of all of its super sets which are elements of the power set, in all the n CEF's This in turn implies that all such super sets are present in the set of intersection of all the n CEF's also, thus making it a CEF A similar reasoning could be made in order to prove that the constructive set of union of all the n CEF's is also a CEF Thus one can see that the structure $(\Phi, \subseteq)$ is a complete lattice The distributivity of this lattice can be proved in the following manner Let us consider three CEF's F_1, F_2 and F_3 decided by $\left\{\Re^{\in}_{F_1}\right\}\square\left\{\Re^{=}_{F_1}\right\}$, $\left\{\Re^{\in}_{F_2}\right\}\square\left\{\Re^{=}_{F_2}\right\}$ and $\left\{\Re^{\in}_{F_3}\right\}\square\left\{\Re^{=}_{F_3}\right\}$ respectively The distributivity is viewed in two ways

(i) $F_1 \cap (F_2 \cup F_3) \simeq (F_1 \cap F_2) \cup (F_1 \cap F_3)$ and

(ii) $F_1 \cup (F_2 \cap F_3) \simeq (F_1 \cup F_2) \cap (F_1 \cup F_3)$

such that the following equalities hold

(i) $$\left\{\Re^{\in}_{F_1 \cap (F_2 \cup F_3)}\right\}\square\left\{\Re^{=}_{F_1 \cap (F_2 \cup F_3)}\right\} \simeq \left\{\Re^{\in}_{(F_1 \cap F_2) \cup (F_1 \cap F_3)}\right\}\square\left\{\Re^{=}_{(F_1 \cap F_2) \cup (F_1 \cap F_3)}\right\}$$

(ii) $$\left\{\Re^{\in}_{F_1 \cap (F_2 \cup F_3)}\right\}\square\left\{\Re^{=}_{F_1 \cap (F_2 \cup F_3)}\right\} \simeq \left\{\Re^{\in}_{(F_1 \cap F_2) \cup (F_1 \cap F_3)}\right\}\square\left\{\Re^{=}_{(F_1 \cap F_2) \cup (F_1 \cap F_3)}\right\}$$

Now let us consider equality (i) Any element of the set in the left hand side of the equality should either be the element of both $\left\{\Re^{\in}_{F_1}\right\}\square\left\{\Re^{=}_{F_1}\right\}$ and $\left\{\Re^{\in}_{F_2}\right\}\square\left\{\Re^{=}_{F_2}\right\}$ or both $\left\{\Re^{\in}_{F_1}\right\}\square\left\{\Re^{\doteq}_{F_1}\right\}$ and $\left\{\Re^{\in}_{F_3}\right\}\square\left\{\Re^{\doteq}_{F_3}\right\}$ This proves equality (i) Equality (ii) can be proved in similar terms using the principle of duality

So, the structure $(\Phi, \subseteq)$ is a complete distributive lattice

As a generalization of distributivity one can realize the following equalities

(i) $$\left\{\Re^{\in}_{F \cup (\bigcap_{i=1}^{n} F_i)}\right\}\square\left\{\Re^{=}_{F \cup (\bigcap_{i=1}^{n} F_i)}\right\} \simeq \left\{\Re^{\in}_{\bigcap_{i=1}^{n} (F \cup F_i)}\right\}\square\left\{\Re^{=}_{\bigcap_{i=1}^{n} (F \cup F_i)}\right\}$$

$$\text{(ii)}\quad \left\{\Re^{\in}_{F\cap(\bigcup_{i=1}^{n} F_i)}\right\}\square\left\{\Re^{=}_{F\cap(\bigcup_{i=1}^{n} F_i)}\right\} \simeq \left\{\Re^{\in}_{\bigcup_{i=1}^{n}(F\cap F_i)}\right\}\square\left\{\Re^{=}_{\bigcup_{i=1}^{n}(F\cap F_i)}\right\}$$

DEFINITION 9 2 1 2 1

Given a base set X_0 with the PEP $\mathcal{P}_0$, one can specify an arbitrary system $\mathcal{P}_\sigma$ of $\mathcal{P}_0$-imbeddable properties by which the process $\left\{\Re^{\in}_{\sigma}\right\}\square\left\{\Re^{=}_{\sigma}\right\}$ would decide a certain collection of subsets of X_0 Now one can specify a filter property $\mathcal{P}_{\bar{\sigma}}$ such that $\left\{\Re^{\in}_{\bar{\sigma}}\right\}\square\left\{\Re^{=}_{\bar{\sigma}}\right\}$ would decide a CEF which contains all the elements of the constructive set decided by the process $\left\{\Re^{\in}_{\sigma}\right\}\square\left\{\Re^{=}_{\sigma}\right\}$ and their corresponding super sets which are the subsets of X_0 . Then the CEF $\left\{\Re^{\in}_{\bar{\sigma}}\right\}\square\left\{\Re^{=}_{\bar{\sigma}}\right\}$ is called the *CEF hull* of the *base* $\left\{\Re^{\in}_{\sigma}\right\}\square\left\{\Re^{=}_{\sigma}\right\}$ and it is the smallest filter that the base could generate

THEOREM 9 2 1 2.2

Let $\left\{\Re^{\in}_{\sigma_1}\right\}\square\left\{\Re^{=}_{\sigma_1}\right\}$ be a constructive subset of another $\left\{\Re^{\in}_{\sigma_2}\right\}\square\left\{\Re^{=}_{\sigma_2}\right\}$ where both are subsets of the base set X_0 Then the corresponding CEF hulls are related by the expression

$$\left\{\Re^{\in}_{\bar{\sigma}_1}\right\}\square\left\{\Re^{=}_{\bar{\sigma}_1}\right\} \subseteq \left\{\Re^{\in}_{\bar{\sigma}_2}\right\}\square\left\{\Re^{=}_{\bar{\sigma}_2}\right\}$$

PROOF

The proof is the direct consequence of the definitions 9.2 1.. and 9 2 1.2 1

DEFINITION 9 2 1 2 2

A CEF or in general any family of constructive sets is said to have *Pairwise Intersection Property* (PIP) if and only if any two elements of the CEF or any two-member subfamily of the family of sets has a nonempty intersection. Similarly, a CEF or a family of constructive sets is said to have *Finite Intersection Property* (FIP) if and only if every finite number of elements of the CEF or every finite-member subfamily of the family of sets has a nonempty intersection.

One can easily verify that if a CEF base $\left\{\Re^{\in}_{\sigma}\right\}\square\left\{\Re^{=}_{\sigma}\right\}$ has either PIP or FIP, then the corresponding CEF hull $\left\{\Re^{\in}_{\bar{\sigma}}\right\}\square\left\{\Re^{\doteq}_{\bar{\sigma}}\right\}$ also has the same property.

DEFINITION 9 2 1 2 3

Let X_0 be a base set and X^k_{ij} be any one of its subsets Then the complement of X^k_{ij} with respect to X_0 is a set decided by $\left\{\Re^{\in}_{X_0 - X^k_{ij}}\right\}\square\left\{\Re^{=}_{X_0 - X^k_{ij}}\right\}$ by virtue of the property $\mathcal{P}^{k'}_{ij'}$, $\mathcal{P}^{k'}_{ij'}$ holds for all the elements of X_0 other than those which form the set X^k_{ij} by satifying the property $\mathcal{P}^k_{ij}$ Now, one can specify a filter property $\mathcal{P}_{\overline{F}}$ ich that the CEF $\left\{\Re^{\in}_{\overline{F}}\right\}\square\left\{\Re^{=}_{\overline{F}}\right\}$ contains strictly either

$\left\{\Re^{\in}_{X^k_{ij}}\right\}\square\left\{\Re^{=}_{X^k_{ij}}\right\}$ or $\left\{\Re^{\in}_{X_0 - X^k_{ij}}\right\}\square\left\{\Re^{=}_{X_0 - X^k_{ij}}\right\}$ in which case it is known as *Universal Filter*

DEFINITION 9 2 1 2.4

Let X_0 be a base set and X_1 its power set Let $\left\{\Re^{\in}_{\sigma}\right\}\square\left\{\Re^{=}_{\sigma}\right\}$ be a CEF base set which consists of a single element from X_1 Then the corresponding CEF hull $\left\{\Re^{\in}_{\overline{\sigma}}\right\}\square\left\{\Re^{=}_{\overline{\sigma}}\right\}$ is known as a *Principal Filter* If the single element of the filter base from X_1 happens to be a single element of X_0 then the corresponding principal filter is a universal filter which is also known as ultra filter of the cartan type

The family of all CEF's over the base set X_0 exhibits a complete distributive lattice $(\Phi, \subseteq)$ where the symbol $\subseteq$ refers to the partial order relation 'coarser than'

EXAMPLE 9 2 1 2 2

Let the base set be $\left\{\Re^{\in}_{X_0}\right\}\square\left\{\Re^{=}_{X_0}\right\}$ = {010,0110,01110} so that the power set is $\left\{\Re^{\in}_{X_1}\right\}\square\left\{\Re^{=}_{X_1}\right\}$ = {{},{010},{0110},{01110},{010,0110},{010,01110},{0110,01110},{010,0110,01110}}.

One can construct 18 CEF's over X_0 [Ref table 9 2 1.2 2] and the lattice Φ formed by these CEF's are shown in figure 9 2 1 2 2

Table 9 2 1 2 2 Constructive extended filters (CEF's) over X_0 = {010,0110,01110}

Sl No	CEF's	CEF bases
1	$\{\Re^{\in}_{F_1}\}\Box\{\Re^{=}_{F_1}\}$ = {{010},{0110},{01110}, {010,0110}{010,01110}, {0110,01110}, {010.0110,01110}}	$\{\Re^{\in}_{\sigma 1}\}\Box\{\Re^{=}_{\sigma 1}\}$ = {{010},{0110},{01110}}
2	$\{\Re^{\in}_{F_2}\}\Box\{\Re^{=}_{F_2}\}$ = {{010},{0110},{010,0110}, {010,01110},{0110,01110}, {010,0110,01110}}	$\{\Re^{\in}_{\sigma 2}\}\Box\{\Re^{=}_{\sigma 2}\}$ = {{010},{0110}}
3	$\{\Re^{\in}_{F_3}\}\Box\{\Re^{=}_{F_3}\}$ = {{010},{01110},{010,0110}, {010,01110},{0110,01110}, {010,0110,01110}}	$\{\Re^{\in}_{\sigma 3}\}\Box\{\Re^{=}_{\sigma 3}\}$ = {{010},{01110}}
4	$\{\Re^{\in}_{F_4}\}\Box\{\Re^{=}_{F_4}\}$ = {{0110},{01110},{010,0110}, {{010,01110},{0110,01110}, {010,0110,01110}}	$\{\Re^{\in}_{\sigma 4}\}\Box\{\Re^{=}_{\sigma 4}\}$ = {{0110},{01110}}
5	$\{\Re^{\in}_{F_5}\}\Box\{\Re^{=}_{F_5}\}$ = {{010},{010,0110},{010,01110}, {0110,01110},{010,0110,01110}}	$\{\Re^{\in}_{\sigma 5}\}\Box\{\Re^{=}_{\sigma 5}\}$ = {{010},{0110,01110}}
6	$\{\Re^{\in}_{F_6}\}\Box\{\Re^{=}_{F_6}\}$ = {{0110},{010.0110},{010,01110}, {0110,01110},{010,0110,01110}}	$\{\Re^{\in}_{\sigma 6}\}\Box\{\Re^{=}_{\sigma 6}\}$ = {{0110},{010,01110}}
7	$\{\Re^{\in}_{F_7}\}\Box\{\Re^{=}_{F_7}\}$ = {{01110}.{010,0110}.{010,01110}, {0110,01110},{010,0110,01110}}	$\{\Re^{\in}_{\sigma 7}\}\Box\{\Re^{=}_{\sigma 7}\}$ = {{01110},{010,0110}}
8	$\{\Re^{\in}_{\bar{F}_8}\}\Box\{\Re^{=}_{\bar{F}_8}\}$ = {{010},{010,0110},{010,01110}, {010,0110.01110}}	$\{\Re^{\in}_{\sigma 8}\}\Box\{\Re^{=}_{\sigma 8}\}$ = {{010}}
9	$\{\Re^{\in}_{\bar{F}_9}\}\Box\{\Re^{=}_{\bar{F}_9}\}$ = {{0110},{010,0110},{0110,01110}, {010,0110,01110}}	$\{\Re^{\in}_{\sigma 9}\}\Box\{\Re^{=}_{\sigma 9}\}$ = {{0110}}
10	$\{\Re^{\in}_{\bar{F}_{10}}\}\Box\{\Re^{=}_{\bar{F}_{10}}\}$ = {{01110},{010,01110}, {0110,01110},{010,0110,01110}}	$\{\Re^{\in}_{\sigma 10}\}\Box\{\Re^{=}_{\sigma 10}\}$ = {{01110}}
11	$\{\Re^{\in}_{\bar{F}_{11}}\}\Box\{\Re^{=}_{\bar{F}_{11}}\}$ = {{010.0110}.{010,01110}, {0110,01110},{010,0110,01110}}	$\{\Re^{\in}_{\sigma 11}\}\Box\{\Re^{=}_{\sigma 11}\}$ = {{010,0110}, {010,01110},{0110,01110}}
12	$\{\Re^{\in}_{F_{12}}\}\Box\{\Re^{=}_{F_{12}}\}$ = {{010,0110},{010,01110}, {010,0110,01110}}	$\{\Re^{\in}_{\sigma 12}\}\Box\{\Re^{=}_{\sigma 12}\}$ = {{010,0110}, {010,01110}}
13	$\{\Re^{\in}_{F_{13}}\}\Box\{\Re^{=}_{F_{13}}\}$ = {{010,0110},{0110,01110}, {010,0110,01110}}	$\{\Re^{\in}_{\sigma 13}\}\Box\{\Re^{\doteq}_{\sigma 13}\}$ = {{010,0110}, {0110,01110}}

contd

14	$\{\Re^{\in}_{F_{14}}\}\square\{\Re^{=}_{F_{14}}\}$ = {{010,01110},{0110,01110}, {010,0110,01110}}	$\{\Re^{\in}_{\sigma 14}\}\square\{\Re^{=}_{\sigma 14}\}$ = {{010,01110}, {0110,01110}}
15	$\{\Re^{\in}_{F_{15}}\}\square\{\Re^{=}_{F_{15}}\}$ = {{010,0110},{010,0110,01110}}	$\{\Re^{\in}_{\sigma 15}\}\square\{\Re^{=}_{\sigma 15}\}$ = {{010,0110}}
16	$\{\Re^{\in}_{F_{16}}\}\square\{\Re^{=}_{F_{16}}\}$ = {{010,01110},{010,0110,01110}}	$\{\Re^{\in}_{\sigma 16}\}\square\{\Re^{=}_{\sigma 16}\}$ = {{010,01110}}
17	$\{\Re^{\in}_{F_{17}}\}\square\{\Re^{=}_{F_{17}}\}$ = {{0110,01110},{010,0110,01110}}	$\{\Re^{\in}_{\sigma 17}\}\square\{\Re^{=}_{\sigma 17}\}$ = {{0110,01110}}
18	$\{\Re^{\in}_{F_{18}}\}\square\{\Re^{=}_{F_{18}}\}$ = {{010,0110,01110}}	$\{\Re^{\in}_{\sigma 18}\}\square\{\Re^{=}_{\sigma 18}\}$ = {{0110,01110}}

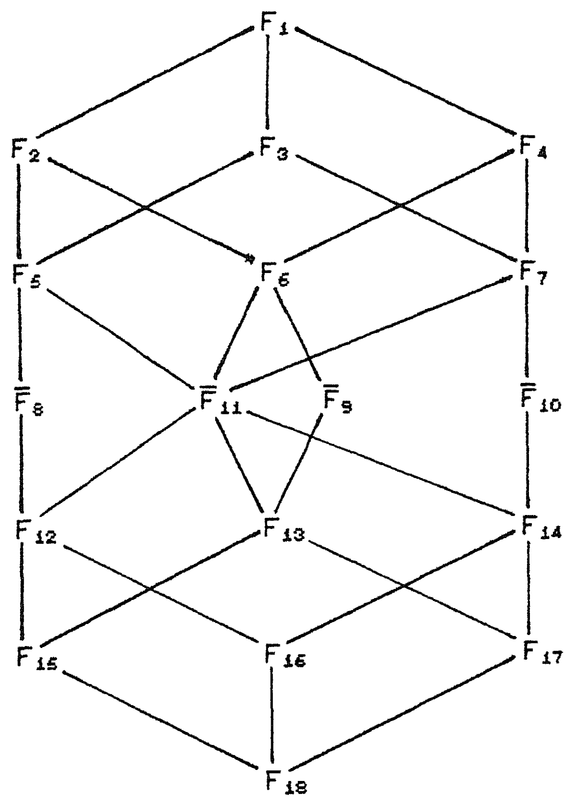


FIGURE 9 2 1.2 2 Lattice formed by the CEF's over X_0 = {010,0110,01110}

9 3 DESCRIPTION OF NORMAL ALGORITHMS IN TERMS OF CONSTRUCTIVE EXTENDED FILTER BASES

In the beginning of subsection 9 2., it was argued that a normal algorithm $\mathcal{N}$ over an alphabet $\mathcal{A}$ is a filter in the sense that it divides the free monoid $\mathcal{A}^*$ into two disjoint sets $\$^{!\mathcal{N}}$ and $\$^{\neg!\mathcal{N}}$ where $\$^{!\mathcal{N}}$ is the set of all strings in $\mathcal{A}^*$ to which $\mathcal{N}$ is sa-definite [Ref subsection 7 2] and $\$^{\neg!\mathcal{N}}$ is its complement with respect to $\mathcal{A}^*$ So the normal algorithmic filter $\mathcal{N}$ is represented by the ordered dichotomy $< \$^{!\mathcal{N}}, \$^{\neg!\mathcal{N}} >$

Here we show how a normal algorithm could be visualized in terms of constructive extended filter bases

With reference to definition 2 2 1., a normal algorithm $\mathcal{N}$ over an alphabet $\mathcal{A}$ consists of an ordered list of substitution formulas of the type $P \longrightarrow Q$ where each substitution formula could be viewed as a map of the type $\mathcal{A}^*P\mathcal{A}^* \longrightarrow \mathcal{A}^*Q\mathcal{A}^*$ The left hand side of the map is a set containing the element P which is a subset of the free monoid $\mathcal{A}^*$ and elements for which P is a factor Similarly the right hand side of the map is a set containing the element Q which is a subset of the free monoid $\mathcal{A}^*$ and elements for which Q is a factor We shall denote both the sets $\mathcal{A}^*P\mathcal{A}^*$ and $\mathcal{A}^*Q\mathcal{A}^*$ by $\hat{P}$ and $\hat{Q}$ respectively Just as the partial order relation 'subset of' indicates the occurrence of a set in another, the partial order relation 'factor of' [Ref Sl No 3 of Table 8 1 1] indicates the occurrence of a string of symbols in another string So, $\hat{P}$ and $\hat{Q}$ could be interpreted as CEF's with their respective bases P and Q Thus a substitution formula $P \longrightarrow Q$ of the normal algorithm $\mathcal{N}$ is viewed as an ordered pair $< P, Q >$ of constructive extended filter bases This amounts to saying that the normal algorithm $\mathcal{N}$ is a totally ordered list of ordered pairs of constructive extended filter bases

Normal algorithms are thus seen to have a direct relevance in all potential applications of Hammer's topological techniques in signal processing

SECTION 10

QUANTIFIABLE MEASURES OF CONSTRUCTIVE EXTENDED FILTERS IN TERMS OF NORMAL ALGORITHMIC OPERATORS

Erlandson introduces three different measures of the filtering capabilities of extended filters (i) the ambiguity of a filter, (ii) the discrimination of a filter and (iii) the resolution of a ilter [43] In addition, he introduces a metric over a space of filters, a measure of how close or far apart two different filters are

Significantly, Erlandson's measures can be given a constructive interpretation as well In fact, we could interpret these measures as normal algorithmic operators mapping a metric lattice of extended filters over a constructive set, into the metric space of constructive real numbers But to describe our interpretation, we need to go into the details of constructive mathematical analysis, starting from the formulation of various constructive numbers, functions and function spaces This would cause a diversion from the main flow of the thesis. So, we describe in what follows, the quantifiable measures for CEF's on similar lines of Erlandson, without loosing the constructive aspects in them

10 1 QUANTIFIABLE MEASURES FOR CONSTRUCTIVE EXTENDED FILTERS

10 1 1 AMBIGUITY OF A CEF PROCESS

Let us take the lattice $(\Phi, \subseteq)$ formed by a family of CEF's over an alphabet $\mathcal{A}$. Ideally, a filter should not accept both an element and its complement, whereas, such a condition is not stipulated in the definition of a CEF So, any CEF that accepts both an element and its complement is called an *ambiguous filter*

We denote the ambiguity measure of a constructive extended filter F_i of k elements by ψ_a and define it as

$$\psi_a = \frac{|\{ X \quad X \subseteq F_1 \text{ and both } X \text{ and its complement } \bar{X} \text{ are in } F_1\}|}{2^{k-1}}$$

Universal filters are non ambiguous The coarsest CEF contained in Φ is the least ambiguous filter and the finest CEF in Φ is the most ambiguous filter.

10 1 2 DISCRIMINATION OF A CEF PROCESS

Discrimination of a CEF, F_1 of k elements, is a measure of the number of elements accepted by it We denote this measure as ψ_d and define it as

$$\psi_d = \frac{|F_1|}{2^{k-1}}$$

The finest CEF is the most discriminating filter and the coarsest CEF is the least discriminating filter

10.1 3 RESOLUTION OF A CEF PROCESS

Resolution is a measure of the smallest size of of the elements accepted by a CEF We define this measure in the following manner

Let F_1 be a CEF and σ_1 be its base Now, we denote the resolution measure of a CEF, F_1 of k elements as ψ_r and define it as $\psi_r = \frac{1}{|\sigma_1|}$

The finest CEF has a resolution 1/k, where, k is the cardinal number of the CEF and the coarsest CEF has a resolution 1

10 1 4 DISTANCE MEASURE BETWEEN TWO CEF'S

Consider two arbitrary CEF's, F_1 and F_j in a lattice of CEF's. Now the distance between these two CEF's is given by

$$\psi_\delta = \frac{|F_1 \cup F_j| - |F_1 \cap F_j|}{|F_i \cup F_j|}$$

EXAMPLE 10 1 1

Let us recall the example 9 2 1 2 2 and characterize all the 18 CEF's based on their Erlandson's measures

Let $\mathcal{A}$ be an alphabet and X_0 = {010,0110,01110} be a base set constructed over $\mathcal{A}$ One can construct 18 extended filters over the set X_0 The complete

distributive lattice formed by these 18 CEF's [Figure 9 2 1 2.2] is once again shown here in figure 10.1 1 along with the distance measures indicated at the appropriate places [NOTE We interpret the word 01/01111111 as the rational number 1/7 A similar interpretation has to be made for any other word of this type]

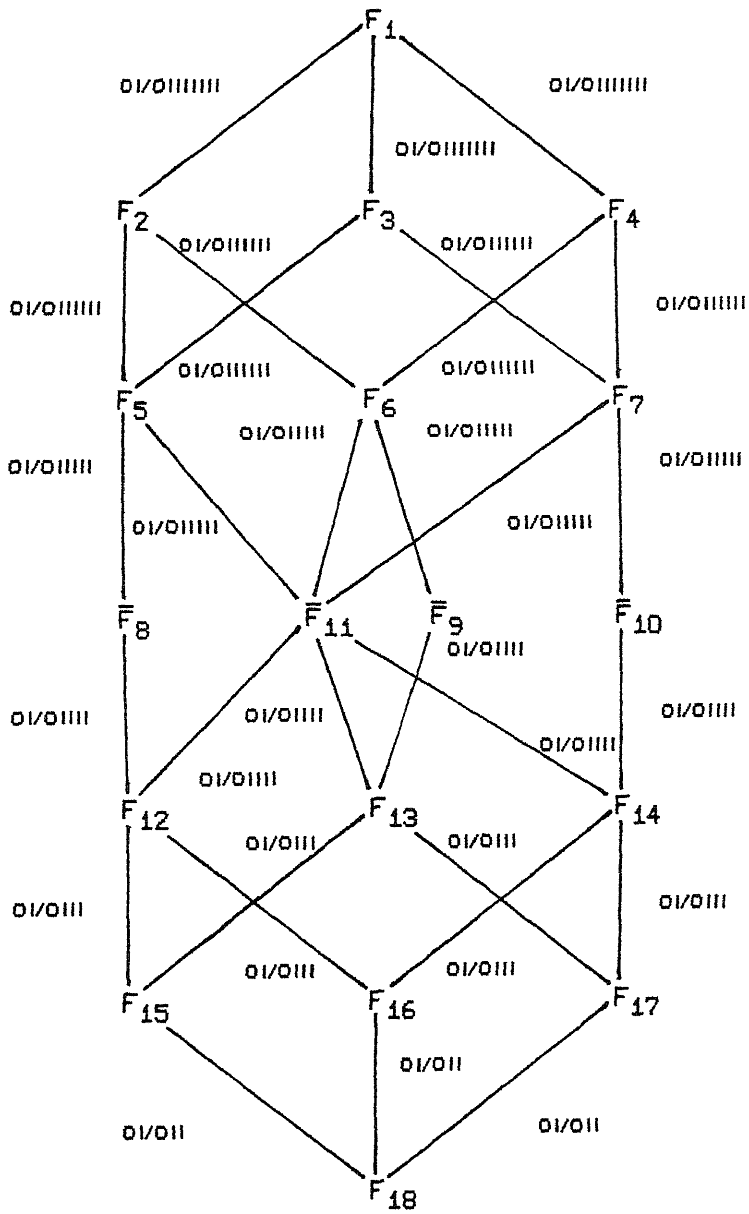


FIGURE 10 1 1 Same as figure 9 2 1 2 2

The remaining measures pertaining to the above example are given in table 10 1 1

Table 10 1 1 Quantifiable measures of CEF's over X_D = (010,0110,01110)

Sl No	$\{\mathcal{N}^{\psi_d}_{F_1}\}$	$\{\mathcal{N}^{\psi_d}_{F_1}\}$	$\{\mathcal{N}^{\psi_d}_{F_i}\}$
1	0111/01111	0111111/01111	01/0111
2	011/01111	011111/01111	01/011
3	011/01111	011111/01111	01/011
4	011/01111	011111/01111	01/011
5	01/01111	01111/01111	01/011
6	01/01111	01111/01111	01/011
7	01/01111	01111/01111	01/011
8	0/01111	0111/01111	01/01
9	0/01111	0111/01111	01/01
10	0/01111	0111/01111	01/01
11	0/01111	0111/01111	01/0111
12	0/01111	0111/01111	01/011
13	0/01111	0111/01111	01/011
14	0/01111	0111/01111	01/011
15	0/01111	011/01111	01/01

contd. .

16	0/01111	011/01111	01/01
17	0/01111	011/01111	01/01
18	0/01111	01/01111	01/01

10.2 COMBINATORIAL THEOREMS INVOLVING QUANTIFIABLE MEASURES OF CEF's

Certain observations were made while computing Erlandson's measures for various CEF's over base sets whose cardinal numbers are less than or equal to three

Those observations are stated here in the form of the following theorems

THEOREM 10 2 1

Let X_0 be the base set whose cardinal number is k. Then the following statements are valid .

(i) The finest filter is the most ambiguous filter over X_0 whose $k \leq 3$ or $k>3$

(ii) The coarsest filter is the least ambiguous filter over X_0 whose $k \leq 3$ or $k>3$

(iii) All the universal filters are nonambiguous filters over X_0 whose $k \leq 3$ or $k>3$

(iv) All the CEF's which are coarser than universal filters over X_0 whose $k \leq 3$ are nonambiguous filters

(v) Ambiguity creeps in certain CEF's which are coarser than universal filters over X_0 whose $k>3$

PROOF

(i) It is obvious that the cardinal number of the power set X_1 of the base set X_0 is 2^k The finest filter contains all the elements of X_1 other than the null set This means that the finest filter allows maximum number of subsets along with their complements too, thus possessing the highest measure of ambiguity

(ii) The coarsest filter contains the base set X_0 as its only element This shows that it is the least ambiguous (nonambiguous) filter

(iii) A universal filter by virtue of its definition satisfies pairwise intersection property This rules out the presence of any pair of elements of a universal filter which are complement to each other So the universal filters are nonambiguous

(iv) Refer to table 10 2 1 All the CEF's F_{12} to F_{18} are nonambiguous

(v) Let us consider X_0 = {010,0110,01110,011110} whose k = 4 One can construct CEF's which are ambiguous and coarser than the universal filters similar to the one given below

$$\left\{\Re^{\in}_{F_1}\right\}\square\left\{\Re^{=}_{F_1}\right\} = \{\{010,0110\},\{01110,011110\},\{010,0110,01110\},\{010,0110,011110\},\{010,01110,011110\},$$
$$\{0110,01110,011110\},\{010,0110,01110,011110\}\}$$

The number of ambiguous filters which are coarser than the universal filters increases when base sets of higher cardinalities are considered

THEOREM 10 2.2

Let X_0 be the base set whose cardinal number is k Then CEF's whose cardinal numbers are greater than 2^{k-1} do not possess pairwise intersection property (PIP)

PROOF

All CEF's whose cardinal numbers are greater than 2^{k-1} are ambiguous and ambiguous filters do not possess PIP

COROLLARY·

All those CEF's which possess PIP would be of cardinalities less than or equal to 2^{k-1} This is the direct consequence of theorem 10 2.2.

THEOREM 10 2 3

The discrimination measure of a family of CEF's over a base set X_0 of cardinality k, ranges from $\frac{1}{2^{k-1}}$ to $\frac{2^k-1}{2^{k-1}}$ in steps of $\frac{1}{2^{k-1}}$

PROOF

As defined earlier, the discrimination ψ_d of a CEF, F_1 is given by $\psi_d = \frac{|F_1|}{2^{k-1}}$ Now the complete distributive lattice formed by the family of CEF's over a given base set of cardinality k, consists of linear chains of CEF's Each linear chain contains 2^{k-1} CEF's which are linked by the relation "just finer than" The distance measure between any two CEF's, of which one is just finer than the other is $\frac{1}{2^{k-1}}$ This is true for all linear chains contained in the lattice Hence the theorem is proved

THEOREM 10 2 4

For base sets with $k \geq 3$, every universal filter which satisfies both FIP and PIP, yields a universal filter of resolution 1/k and which satisfies only PIP, when the base element of the former is replaced by its complement with respect to X_0

PROOF

The universal filters over a base set X_0 , which satisfy both FIP and PIP are ultra filters of cartan type whose single element bases consist of elements from X_0 Let us consider one such filter $\tilde{F}_1$ whose base consists of any one of the k elements of X_0 By virtue of the definition 9 2 1 1 1, the remaining $(2^{k-1}-1)$ elements of $\tilde{F}_1$ must contain this base element If this base element is replaced by its complement set which is a subset of X_0 consisting of (k-1) elements of X_0 , then the finite intersection property does not hold for the resulting filter On the other hand, the complement set has a nonempty intersection with each of the $(2^{k-1}-1)$ elements of $\tilde{F}_1$ So, the resulting filter is a universal filter which satisfies only PIP Let us denote this resulting filter by $\tilde{F}_{1c}$ The base of $\tilde{F}_{1c}$ will then consist of k elements, in which (k-1) elements would be the super sets of cardinality 2 of the base element of $\tilde{F}_1$ and the remaining one element would be the complement set of the base element of $\tilde{F}_1$ For a base set X_0 with k=1 there is only one universal filter and the question of complement set of its base does not arise

at all For a base set X_0 with k=2 there are two universal filters whose bases are complements to each other So the question of getting a universal filter of the type $\tilde{F}_{1_C}$ does not arise at all For a base set X_0 with k=3 there are three universal filters and only one universal filter of the type $\tilde{F}_{1_C}$ would result when any of the bases of the three $\tilde{F}_1$'s is complemented [Ref table 9 2 1 2.2] For a base set X_0 with k>3 each of the k universal filters of the cartan type ultra filters would yield a unique and independent universal filter of the type $\tilde{F}_{1_C}$ with the resolution 1/k when its base element is complemented

The following theorem due to Erlandson, characterizes a certain kind of universal filters in terms of 'resolution'

THEOREM 10 2 5 [39]

Given a base set X_0 with $k \geq 3$ and k being an odd number, one can construct a universal filter $\overline{F}_1$ with ${}^kC_{[k/2]}$ elements

[NOTE [k/2] denotes the next integer greater than the result of dividing k by 2. Similarly (k/2) denotes the next integer less than the result of dividing k by 2]

CONCLUSIONS AND PERSPECTIVES

The results presented in this thesis are the outcome of an effort to build a constructive logical theory for nonnumerical signal processing

In our opinion, the constructive approach to nonnumerical signal processing adopted in this thesis, has turned out to be successful in the sense that several central notions of traditional numerical signal processing could also be interpreted within the framework of constructive matl ·matical logic and a few new notions have emerged together with results concerning them

The work presented here nevertheless marks only a beginning and, besides some of the loose ends indicated at several places in the text, there are a number of interesting problems and ideas that can be suggested for future study. In summary, these are as follows

1 The idea of using constructive extended filters in the topological processing of signals and images would seem to be of considerable promise

2 The concept of M-grammar, which has been developed in section-5, could be studied in the light of *Selective Substitution Grammar*, which was introduced by Rozenberg [29], [30], with an intention to provide a general formal definition of a rewriting system This may open up possibilities of using M-grammar in multidimensional image pattern generation similar to the use of Lindenmayer rewriting systems

3 The possibility may be explored for the use of the notion of a constructive set, which has been developed in section-9, in morphological image processing. It is to be noted that morphological operations like *erosion* and *dilation* are set theoretic operations that would seem to lend themselves well to a constructive adaptation

4 Computable analysis of Boolean functions by means of normal algorithms is another interesting area for potential research [NOTE. A Boolean function of n variables is a function of an n-dimensional Boolean vector An n-dimensional Boolean vector is a word of length n from the alphabet $\mathcal{A}_0$ = { 0 1 }] The material provided in the first part of this thesis would be of help in constructing normal algorithms that compute Boolean functions

5 Finally, an important step that one could take is to construct efficient norma algorithms for various requirements so that a suitable environmen could be developed for carrying out further research and exploring the possibility of hybridising numeric and nonnumeric signal processing

As a closing remark, it may be added that the results presented in this thesis are likely to find their use in areas such as those of Image and signal processing, Control systems, Logic and computing, Formal languages and automata, Symbolic artificial intelligence and Communication networks

APPENDIX - 1 [A 1]

CYCLIC SHIFTING SUBROUTINE

LISTING OF A SUBROUTINE

DESIGNED TO IMPLEMENT DESIRED NUMBER OF CYCLIC SHIFTS

IN ANY ARBITRARILY LONG STRING OF SYMBOLS FROM ANY GIVEN ALPHABET

(ALGORITHM $\mathfrak{R}^{CS}$)

```
C*********************************************************************
C       BY CYCLIC SHIFTING WE UNERSTAND THE SHIFTING OF THE RIGHT MOST
C       SYMBOL IN A GIVEN STRING OF SYMBOLS TO ITS EXTREME LEFT  THIS
C       PROGRAM CYSHFT IMPLEMENTS DESIRED NUMBER OF SUCH CYCLIC
C       SHIFTS IN A GIVEN STRING
C*********************************************************************
        DIMENSION ITEMP(4096),ICH(4096)
        OPEN(UNIT=1,FILE='CYSHFT DAT',STATUS='NEW')
        WRITE(1,79)
        WRITE(*,79)
79      FORMAT(10X,'GIVE THE NO OF CHARACTERS IN THE INPUT STRING ')
        READ(*,*)NCHAR
        WRITE(1,*)NCHAR
        WRITE(*,81)
        WRITE(1,81)
81      FORMAT(10X,'GIVE THE INPUT STRING '/10X,'[ Press <return> only
       1 after the complete string is given ]')
        READ(*,'(100A1)')(ITEMP(I),I=1,NCHAR)
        WRITE(1,92)(ITEMP(I),I=1,NCHAR)
92      FORMAT(/10X,100A1)
        WRITE(*,82)
        WRITE(1,82)
82      FORMAT(10X,'SPECIFY NUMBER OF CYCLIC SHIFTS.')
        READ(*.*)NUM
        WRITE(1,*)NUM
        WRITE(1,*)
        DO 230 IT=1,NUM
175     KNT=3
        I1=1
        I2=2
        I3=2
        I4=2
        IJ1=0
        ICOUNT=0
        IF(NCHAR LT 4)GO TO 123
30      N1=1
        N2=N1+KNT
28      IJ=IJ1
        DO 20 I=N1,N2,I3
        IF(ITEMP(I+I1).NE IJ)GO TO 15
        IJ=IJ+I4
```

```
20      CONTINUE
        GO TO(16,16,31,7,7,8,16,8,8,16),ICOUNT+1
31      DO 35 IS=1,3
        IF(ITEMP(N2).NE.(IS-1))GO TO 35
        GO TO 15
35      CONTINUE
16      DO 10 I=1,I2
        IATEMP=ITEMP(N1)
        DO 5 J=N1,N2-1
        ITEMP(J)=ITEMP(J+1)
5       CONTINUE
        ITEMP(N2)=IATEMP
10      CONTINUE
        IF(ICOUNT.EQ.2)GO TO 190
1       DO 300 I=1,NCHAR
        IF(ITEMP(I).NE.0)GO TO 11
        ICH(I)='0'
        GO TO 300
11      IF(ITEMP(I).NE.1)GO TO 12
        ICH(I)='1'
        GO TO 300
12      IF(ITEMP(I).NE.2)GO TO 13
        ICH(I)='2'
        GO TO 300
13      ICH(I)=ITEMP(I)
300     CONTINUE
        WRITE(1,93)ICOUNT,(ICH(I),I=1,NCHAR)
        WRITE(*,93)ICOUNT,(ICH(I),I=1,NCHAR)
93      FORMAT(2X.I4,5X,65A1)
        IF(ICOUNT.NE.8)GO TO 175
        GO TO 230
15      N1=N1+1
        N2=N2+1
        IF(N1.LE.(NCHAR-KNT))GO TO 28
123     ICOUNT=ICOUNT+1
        GO TO (100,110,120,130,140,150,160,170,180),ICOUNT
100     KNT=2
        I1=0
        I2=1
        IF(NCHAR.GE.3)GO TO 30
        GO TO 123
110     KNT=1
        IJ1=1
        IF(NCHAR.GE.2)GO TO 30
        GO TO 123
120     IJ1=0
        I3=1
        I4=0
        IF(NCHAR.GE.2)GO TO 30
        ICOUNT=ICOUNT+1
130     IJ1=1
        IF(NCHAR.GE.2)GO TO 30
140     I3=1
        I4=-2
```

```
      IJ1=2
      IF(NCHAR.GE.2)GO TO 30
      ICOUNT=ICOUNT+1
150   I3=2
      IF(NCHAR.GE.2)GO TO 30
      ICOUNT=ICOUNT+1
160   IJ1=1
      I3=1
      I4=-1
      IF(NCHAR.GE.2)GO TO 30
      ICOUNT=ICOUNT+1
170   KNT=0
      IJ1=2
      GO TO 30
180   N1=1
190   NCHAR=NCHAR+1
      DO 250 I=N1,NCHAR-1
      ITEMP(NCHAR-(I-N1))=ITEMP(NCHAR-(I-N1)-1)
250   CONTINUE
      ITEMP(N1)=0
      GO TO 1
7     ITEMP(N1)=IJ1+1
8     DO 135 J=N2,NCHAR-1
      ITEMP(J)=ITEMP(J+1)
135   CONTINUE
      NCHAR=NCHAR-1
      GO TO 1
230   CONTINUE
      STOP
      END
C**********************************************************************
```

APPENDIX - 2. [A.2]

LINEAR CONVOLUTION SUBROUTINE

LISTING OF A SUBROUTINE

DESIGNED TO COMPUTE LINEAR CONVOLUTION OF

NONNEGATIVE INTEGER SEQUENCES OF ARBITRARY LENGTHS

(ALGORITHM: $\Re^{con}$)

```
C*********************************************************************
C       THIS PROGRAM LINCON IMPLEMENTS THE LINEAR CONVOLUTION OF ANY TWO
C       NON-NEGATIVE INTEGER SEQUENCES OF ARBITRARY LENGTHS PURELY BY
C       MANIPULATING THEIR CORRESPONDING SYMBOLIC REPRESENTATIONS.
C*********************************************************************
        CHARACTER*1 ISYM(5),INUM(10)
        CHARACTER*10 INFIL
        DIMENSION ITEMP(4096),ICH(4096)
        DATA ISYM/'0',',',';','I','*',' '/
        INUM(1)=CHAR(224)
        INUM(2)=CHAR(225)
        INUM(3)=CHAR(226)
        INUM(5)='a'
        INUM(6)='b'
        INUM(7)=CHAR(235)
        INUM(8)=CHAR(229)
        INUM(9)=CHAR(231)
        INUM(10)=CHAR(254)
        WRITE(*,*)' SPECIFY OUTPUT FILE NAME :'
        READ(*,'(A)')INFIL
        OPEN(UNIT=1,FILE=INFIL,STATUS='NEW')
        NCHAR=1
        IFLAG=1
        ITEMP(NCHAR)=9
        IATEMP=ISYM(2)
        WRITE(*,81)
        WRITE(1,81)
81      FORMAT(10X,'GIVE THE LENGTHS OF THE TWO DATA SEQUENCES'/10X,
      1 'TO BE CONVOLVED:')
        READ(*,*)N1,N2
        WRITE(1,*)N1,N2
        IDIFF=IABS(N1-N2)
        IMIN=MIN0(N1,N2)
        WRITE(*,82)
        WRITE(1,82)
82      FORMAT(/10X,'GIVE THE FIRST SEQUENCE:')
901     DO 800 I=1,N1
        READ(*,*)ICH(I)
        NCHAR=NCHAR+1
        IF(ICH(I).NE.0)GO TO 805
        ITEMP(NCHAR)=ISYM(1)
```

```
      NCHAR=NCHAR+1
      GO TO 802
805   DO 801 IT=1,ICH(I)
      ITEMP(NCHAR)=ISYM(3)
      NCHAR=NCHAR+1
801   CONTINUE
802   ITEMP(NCHAR)=IATEMP
800   CONTINUE
      WRITE(*,2020)IFLAG,(ICH(I),I=1,N1)
      WRITE(1,2020)IFLAG,(ICH(I),I=1,N1)
2020  FORMAT(/15X,'SEQUENCE No.',I2,/(/15X,10I4))
      IFLAG=IFLAG+1
      IF(IFLAG.GT.2)GO TO 900
      ITEMP(NCHAR)=3
      IATEMP=9
      N1=N2
      WRITE(*,83)
      WRITE(1,83)
83    FORMAT(/10X,'GIVE THE SECOND SEQUENCE:')
      GO TO 901
900   WRITE(1,*)
      WRITE(1,*)'    FORMULA   ELEMENTARY'
      WRITE(1,*)'    NUMBER    TRANSFORMATIONS'
      WRITE(1,*)'    -------   ---------------'
      WRITE(1,*)
      WRITE(1,*)'              The pre-convolution string is:'
      NCHAR=NCHAR-1
      ITRANS=-1
      GO TO 1
30    N1=1
      N2=N1+KNT
28    IJ=IJ1
      DO 20 I=N1,N2,I3
      IF(ITEMP(I+I1).NE.IJ)GO TO 15
      IJ=IJ+I4
20    CONTINUE
      IF(ICOUNT.GT.90)GO TO 714
      IF(ICOUNT.GT.75)GO TO 712
      IF(ICOUNT.GT 60)GO TO 716
      IF(ICOUNT.GT.45)GO TO 710
      IF(ICOUNT.GT.30)GO TO 705
      IF(ICOUNT.GT.15)GO TO 702
      GO TO(16,480,485,490,16,31,7,7,8,31,16,8,16,500,503,504),
     1 ICOUNT+1
702   GO TO(8,8,116,16,31,8,8,550,553,555,560,565,37,570,575),IKOUNT
705   GO TO(8,565,580,585,590,595,39,525,37,37,37,37,37,
     1 580,39),IKOUNT
710   GO TO(530,532,535,535,540,545,605,610,615,620,625,630,635,8,230
     1 ),IKOUNT
716   GO TO(16,480,485,490,16,31,7,7,8,31,16,8,16,500,503),IKOUNT
712   GO TO(504,8,31,371,391,499,101,102,16,103,501,230,230,
     1 230,230),IKOUNT
714   GO TO(31,509,37,636,616,509,37,637,37,8,230),IKOUNT
39    IV=1
```

```
37      N2=N2+1
        IF(N2.GT.NCHAR)GO TO 15
31      DO 35 IS=1,15
        IF(ITEMP(N2).NE.(IS-1))GO TO 35
        GO TO 15
35      CONTINUE
        IF(ICOUNT.LE.65)GO TO 2000
        IF(ICOUNT.GT.80)GO TO 2001
        IKT=ICOUNT-65
        GO TO(16,230,230,230,16,230,230,230,8,230,230,230,
      1 16,230,230),IKT
2001    IKT=ICOUNT-80
480     IF(ITEMP(N1).NE.9)GO TO 15
        GO TO 16
485     IF(ITEMP(N1+2).NE.9)GO TO 15
        GO TO 16
490     IF(ITEMP(N1+1).NE.9)GO TO 15
        GO TO 16
501     IF(ITEMP(N1).NE.ISYM(2))GO TO 15
        N2=N2+1
        GO TO 31
500     N2=N1
        GO TO 31
503     IF(ITEMP(N1).NE.9)GO TO 15
504     N2=N1
        GO TO 8
499     IF(ITEMP(N1+2).NE.ISYM(2))GO TO 15
        IF(ITEMP(N1).NE.ISYM(3))GO TO 15
506     ITEMP(N1+2)=4
        GO TO 1
508     ITEMP(N1)=3
        ITEMP(N1+1)=ISYM(2)
        N2=N1-1
        IZ=1
        GO TO 538
511     ITEMP(N1)=ISYM(2)
        ITEMP(N1+1)=1
        GO TO 1
509     IF((N2+1).GT.NCHAR)GO TO 15
        IF(ITEMP(N2+1).NE.ISYM(2))GO TO 15
        IF(ICOUNT.EQ.96)GO TO 618
        IF(ICOUNT.EQ.97)GO TO 618
617     DO 136  I=1,2
        DO 137 J=N1,NCHAR-1
        ITEMP(J)=ITEMP(J+1)
137     CONTINUE
        NCHAR=NCHAR-1
136     CONTINUE
        ITEMP(N1)=9
        IF(ICOUNT.EQ.95)ITEMP(N1)=3
        GO TO 1
616     IF(ITEMP(N1+2).NE.9)GO TO 15
        GO TO 617
618     ITEMP(N1)=3
```

```
      ITEMP(N1+1)=6
      GO TO 1
525   IF(ITEMP(N1).NE.9)GO TO 15
      ITEMP(N2)=9
      N2=N2+1
      GO TO 8
530   IF(ITEMP(N2+1).NE.ISYM(3))GO TO 15
      N2=N2+1
      GO TO 8
532   IF(ITEMP(N2+1).NE.ISYM(1))GO TO 15
      N2=N2+1
      GO TO 8
535   IF(ITEMP(N2+1).NE.5)GO TO 15
      IF(ICOUNT.EQ.48)GO TO 112
538   NCHAR=NCHAR+1
      K=N2+1
      DO 125 J=K,NCHAR-1
      ITEMP(NCHAR-(J-K))=ITEMP(NCHAR-(J-K)-1)
125   CONTINUE
      IF(ICOUNT.EQ.86)GO TO 514
      IF(ICOUNT.NE.27)GO TO 529
      ITEMP(N2+1)=5
      GO TO 1
529   IF(ICOUNT.NE.26)GO TO 537
542   ITEMP(N2+1)=4
      GO TO 1
537   IF(ICOUNT.NE.23)GO TO 539
514   IZ=IZ+1
      IF(IZ.LE.2)GO TO 538
      IF(ICOUNT.EQ.86)GO TO 511
      ITEMP(N1)=ISYM(1)
      ITEMP(N1+1)=1
      GO TO 1
539   IF(ICOUNT.NE.24)GO TO 541
      GO TO 542
541   ITEMP(N2+1)=6
      GO TO 1
540   IF(ITEMP(N2+1).NE.4)GO TO 15
      GO TO 541
545   IF(ITEMP(N2+1).NE.7)GO TO 15
      GO TO 541
550   I3=2
      DO 707 I=N1,N2,I3
      IF(ITEMP(I).NE.ISYM(I-(N1-1)))GO TO 15
707   CONTINUE
      ITEMP(N1)=3
      ITEMP(N1+1)=ISYM(1)
      ITEMP(N1+2)=ISYM(3)
      N2=N1-1
      IZ=1
      GO TO 538
553   IF(ITEMP(N1).NE.ISYM(1))GO TO 15
      IF(ITEMP(N1+2).NE.ISYM(1))GO TO 15
      GO TO 638
```

```
555   IF(ITEMP(N1).NE.ISYM(2))GO TO 15
      DO 531 I=1,2
      NCHAR=NCHAR+1
      K=N2+1
      DO 126 J=K,NCHAR-1
      ITEMP(NCHAR-(J-K))=ITEMP(NCHAR-(J-K)-1)
126   CONTINUE
      N2=N2+1
531   CONTINUE
      ITEMP(N1+1)=1
      ITEMP(N1+2)=3
      ITEMP(N1+3)=ISYM(2)
      GO TO 1
560   IF(ITEMP(N1).NE.ISYM(3))GO TO 15
      IF(ITEMP(N1+2).NE.ISYM(1))GO TO 15
564   N2=N2-1
      GO TO 538
565   IF(ITEMP(N2).NE.ISYM(3))GO TO 15
      GO TO 16
570   IF((N1+1).GT.NCHAR)GO TO 15
      IF(ITEMP(N1+1).NE.ISYM(1))GO TO 15
      N1=N1+1
      GO TO 16
575   IF((N1+2).GT.NCHAR)GO TO 15
      DO 22 I=1,2
      IF(ITEMP(N1+I).NE.ISYM(1))GO TO 15
22    CONTINUE
      IF(ICOUNT.EQ.24)GO TO 638
      GO TO 8
580   I6=9
581   IF((N1+2).GT.NCHAR)GO TO 15
      IF(ITEMP(N1+2).NE.I6)GO TO 15
      N2=N2+1
      GO TO 16
585   I6=4
      GO TO 581
590   I6=5
      GO TO 581
595   I6=7
      GO TO 581
605   IF(ITEMP(N1+2).NE.9)GO TO 15
      N2=N2+1
708   IX=3
      DO 718 I=N1,N2
      ITEMP(I)=IX
      IX=IX+3
718   CONTINUE
      GO TO 1
610   IF(ITEMP(N1+2).NE.ISYM(2))GO TO 15
611   ITEMP(N1)=3
      ITEMP(N1+1)=7
      GO TO 1
615   IF(ITEMP(N1+2).NE.5)GO TO 15
      GO TO 708
```

```
620   IF(ITEMP(N1+2).NE.ISYM(3))GO TO 15
      ITEMP(N1)=3
      ITEMP(N1+1)=4
      GO TO 1
625   IF(ITEMP(N1+2).NE.ISYM(1))GO TO 15
      GO TO 611
630   IF(ITEMP(N1+3).NE.ISYM(1))GO TO 15
      ITEMP(N1)=3
      ITEMP(N1+1)=ISYM(1)
      GO TO 1
635   IF(ITEMP(N1+2).NE.6)GO TO 15
636   ITEMP(N1)=3
      ITEMP(N1+1)=8
      IF(ICOUNT.EQ.94)ITEMP(N1+1)=ISYM(2)
      GO TO 1
637   N2=N2+1
      IF(ITEMP(N2).NE.6)GO TO 15
      ITEMP(N1)=3
      ITEMP(N1+1)=5
      GO TO 8
640   ITEMP(N1)=ISYM(3)
      GO TO 1
112   IF(ITEMP(N2+2).NE.4)GO TO 15
      DO 118 I=1,2
      ITEMP(N2+I)=ITEMP(N2+I)+1
118   CONTINUE
      GO TO 1
638   N2=N2-1
      GO TO 538
116   ITEMP(N1)=ISYM(3)
      ITEMP(N1+1)=8
      GO TO 1
36    N1=1
      N2=N1+KNT
27    IJ=IJ3
      DO 24 I=1,2
      IATEMP=ISYM(IJ)
      IF(ITEMP(N1+I-1).NE.IATEMP)GO TO 115
      IJ=IJ+I4
24    CONTINUE
      GO TO 8
115   N1=N1+1
      N2=N2+1
      IF(N1.LE.(NCHAR-KNT))GO TO 27
      GO TO 123
101   IF(ITEMP(N1).NE.ISYM(2))GOTO 15
      IF(ITEMP(N1+2).NE.ISYM(2))GOTO 15
      ITEMP(N1+2)=4
      GOTO 8
102   IF(ITEMP(N1).NE.ISYM(2))GO TO 15
      ITEMP(N1+2)=8
      GO TO 1
103   IF(ITEMP(N1).NE.8)GO TO 15
      ITEMP(N1+1)=9
```

```
      ITEMP(N1+2)=8
      GOTO 1
391   IF((N1+2).GT.NCHAR)GOTO 15
      DO 392 I=1,2
      DO 390 IS=1,10
      IF(ITEMP(N1+I).NE.(IS-1))GO TO 390
      GO TO 15
390   CONTINUE
392   CONTINUE
      N2=N1+2
      GO TO 16
371   IF((N1+2).GT.NCHAR)GO TO 15
      DO 394 I=1,10
      IF(ITEMP(N1+1).NE.(I-1))GO TO 394
      GO TO 15
394   CONTINUE
      IF(ITEMP(N1+2).NE.9)GO TO 15
      N2=N1+2
      GO TO 16
C***********************************************************************
C     TO DISPLAY THE CONVOLVED OUTPUT STRING IN DECIMAL FORM:
2300  IFLAG=1
      ICH(IFLAG)=0
      DO 2400 I=1,NCHAR
      IF(ITEMP(I).NE.ISYM(3))GOTO 2500
      ICH(IFLAG)=ICH(IFLAG)+1
      GOTO 2400
2500  IF(ITEMP(I).EQ.ISYM(1))GOTO 2400
      IFLAG=IFLAG+1
      ICH(IFLAG)=0
2400  CONTINUE
      WRITE(*,2010)(ICH(I),I=1,IFLAG)
      WRITE(1,2010)(ICH(I),I=1,IFLAG)
2010  FORMAT(/15X,'THE CONVOLVED OUTPUT STRING IS:',/(/15X,10I4))
C***********************************************************************
230   STOP
      END
```

APPENDIX - 3. [A.3]

ON CONSTRUCTIVE MATHEMATICS

The adjective *constructive* is used here only in the following sense: the presence of this adjective in the term for a concept, the name of a method, the name of a branch of mathematics, etc., will signify that the indicated concept, method, branch of mathematics, etc., belongs to the *constructive approach* to mathematics.

Markov's constructive approach to mathematics is characterizd by the following features:

(i) In all mathematical theories belonging to this approach, only *constructive objects* (i.e., words from alphabets) figure as objects of study.

(ii) In the study of constructive objects, one is permitted to make use of the idealization of *the abstraction of potential realizability* (i.e., the abstraction from the *real* limits of our constructive possibilities, imposed by the limited character of our lives in space and time), but the use of the abstraction of actual infinity is completely forbidden.

(iii) In accordance with the type of objects of study and the abstraction of potential realizability as a meaningful basis for the construction of mathematical theories one has to take certain constructive interpretation of mathematical judgements.

In what follows, we provide the essential features of the constructive interpretation of mathematical judgements.

Every computing process has to be carried out on the basis of some symbolism of one's choice. In working with any symbolism, one has to initially provide some signs as *elementary signs*. Elementary signs are also called *letters*. A list of such

nonrepeating letters is called *alphabet*. Given an alphabet, one can construct a new alphabet by adding a new letter to the list of letters corresponding to the given alphabet. Given a word from an alphabet, one can construct a new word by concatenating a letter from the alphabet to the given word. A mathematical object (word) is called computable only when it is computable by an algorithm called *normal algorithm*. In other words, a computing process is to be understood as the transformation of a word from an alphabet to another from the same alphabet or its extension, by means of a normal algorithm, constructed ove the alphabet. A normal algorithm is a totally ordered list of semi-Thue type production formulas. These formulas are also known as *substitution formulas*.

A normal algorithm can be completely transcribed (coded) by a word from a two-lettered alphabet. Though the method of coding a normal algorithm is unique as per Markov's *transcription theorem*, there is no unique way of coding it in a two-lettered alphabet. However, an efficient coding of a normal algorithm would be to represent it by the word of the shortest length. The shortest word that describes a normal algorithm is called the *complexity* of the normal algorithm. So, by complexity of a normal algorithm, we actually mean the volume of the program specifying the normal algorithm.

With the idea of describing (normal) algorithms and constructive mathematical logic within the framework of a single theory, Markov developed a system of languages known as the *heirarchical semantic system*. Every logico mathematical language belonging to this semantic system provides the respective rules of writing assertions/propositions about the words from certain basic alphabets and the normal algorithms on these alphabets. This system of languages, denoted as Я_{α}: (α = 0,1,2,3,4,.... ω, ωI), has been built in such a way that the meaning of the formulae of one level is defined in terms of the objects of the previous level.

The main feature of this heirarchical semantic system is expressed in a

theorem of Markov on the completeness of the classical predicate calculus relative to the semantics in constructive logic [66]. In other words, most of the mathematical theories developed within the framework of classical logic have their corresponding constructive analogues within this system. This does not assert that all the rules of deduction of classical logic are acceptable in constructive mathematics. For example, unlike the interpretation given in the classical logic, the notion of *implication* ($A \supset B$) is interpreted in a particular language of this system as a statement of the assertion that B can be inferred from the premise A by means of a semiformal theory of that language. Informally, the implication ($A \supset B$) expresses the feasibility of a construction p such that if q is an arbitrary construction asserting A, then p and q together allow us to look for a construction asserting B.

On the lines of Shanin, it is not necessary in constructing various mathematical theories within the framework of the constructive approach in mathematics, to carry out the proof of each new theorem by applying an algorithm to the corresponding formula for its validity. On the other hand, one can prove a theorem with the help of the basic rules of logical deduction of a particular language in which the corresponding formula is written.

As regards the notion of *constructive sets*, the term *set* is understood as a synonym of the term *condition with one parameter*, i.e., a constructive logical formula in a particular language, with a single free variable. Various operations on sets and various propositions about sets are the corresponding constructive operations on formulas and propositions about formulas. In this manner, the basic logico mathematical languages allow sufficiently broad possibilities for defining *sets*.

In constructive mathematics, the notion of a *number* is understood as the normal algorithm that generates it. So, various operations on numbers are the

corresponding constructive operations on certain normal algorithms. Choosing a constructive analogue of the concept of a real number used in classical mathematics was one of the major problems in the initial development of construtive mathematical anlaysis. Presently a number of definitions of the concept of a constructive real number are available, among which the following is the definition due to Markov and Shanin.

Let $\mathcal{A}_0$ = { 0 I } be an alphabet and $\Diamond$ be a symbol not in $\mathcal{A}_1$ such that $\mathcal{A}$ = {0 I $\Diamond$} is their union. Then the word P$\Diamond$Q is defined as a constructive real number, where P and Q are the transcriptions (words from $\mathcal{A}_0$) corresponding to two specific normal algorithms, say, $\mathfrak{U}$ and $\mathfrak{B}$. The normal algorithm $\mathfrak{U}$ gives the sequence of rational approximating values for the given constructive real number and $\mathfrak{B}$ is a regulator of convergence in itself of $\mathfrak{U}$, i.e., an algorithm computing for any natural number n, the subscript, beginning with which the absolute value of the difference between terms of the sequence of rational approximating values of the given constructive real number is less than 2^{-n}.

The concept of a constructive function f of m real variables is an algorithm of the type $\mathbf{R}^m \longrightarrow \mathbf{R}$ where $\mathbf{R}$ is the system of constructive real numbers.

In this formalism, the constructive differential and integral calculus are very much similar to those of the traditional theories.

Thus, we observe that the entire formulation of the constructive mathematical analysis in this manner, is based on the use of normal algorithms in describing various essential concepts.

REFERENCES

STANDARD BOOKS

[1] Aberth, O., Computable Analysis, McGraw Hill, New York, 1980.

[2] Arbib, M.A., Algebraic Theory of Machines, Languages and Semigroups, Academic Press, New York, 1968.

[3] Barwise, Jon, Handbook of Mathematical Logic, North-Holland Publishing Company, 1977.

[4] Beeson, M.J., Foundations of Constructive Mathematics, Springer Verlag, Heidelberg, 1985.

[5] Benthem, J.F.A.K.V., The Logic of Time, D. Reidel, London, 1983.

[6] Berstel, Jean and Perrin, Dominique, Theory of Codes, Academic Press Inc., 1985.

[7] Bishop, Errett, Foundations of Constructive Analysis, McGraw Hill, 1967.

[8] Bridges, D.S., Constructive Functional Analysis, Pitman, London, 1979.

[9] Clifford and Preston, The Algebraic Theory of Semigroups, American Mathematical Society, Providence, 1961.

[10] Copi, I.M., Symbolic Logic, MacMillan Publishing Company, 1979.

[11] Dalen, D.V. Logic and Structure, Springer Verlag, Heidelberg, 1980.

[12] Davis, Martin, Applied Nonstandard Analysis, John Wiley, 1976.

[13] Ershov, Yu.L., and Palyutin, E.A., Mathematical Logic, Mir Publishers, Moscow, 1984.

[14] Fitting, Melvin, Computability Theory, Semantics and Logic Programming Oxford University Press, 1987.

[15] Giardina, C.R., and Dougherty, E.R., Morphological Methods in Image and Signal Processing, Prentice Hall Inc., 1988.

[16] Gleason, A.M., Fundamentals of Abstract Analysis, Addison Wesley, 1966.

[17] Griswold, R.E., Poage, J.F., and Polonsky, I.P., The SNOBOL4 Programming Language, Prentice Hall, 1971.

[18] Grzegorczyk, A., An Outline of Mathematical Logic, D. Reidel Pub. Co., 1974.

[19] Hofstadter, D.R., Gödel, Escher and Bach: An Eternal Golden Braid, Penguin Basic Books Inc., 1979.

[20] Kushner, B.K., Lectures on Constructive Mathematical Analysis, AMS, Providence, Rhode Island, Vol 60, 1984.

[21] Lallement Gerard, Semigroups and Combinatorial Applications, John Wiley and Sons, 1979.

[22] Lothaire, M., Combinatorics on Words, Addison Wesley Pub. Co., 1983.

[23] Markov, A.A., Theory of Algorithms, The Israel Program for Scientific Translations, 1961.

[24] Mesarovic, M.D. and Takahara. Y., General System Theory: Mathematical Foundations, Academic Press, New York, 1975.

[25] Mittelstaedt, Peter, Quantum Logic, D. Reidel Publishing Company.

[26] Pin, J.E., Varieties of Formal Languages, Plenum Press, New York, 1986.

[27] Reichenbach, Hans, The Direction of Time, University of California Press, 1956.

[28] Robinson, A., Introduction to Model Theory and to the Metamathematics of Algebra, North-Holland Pub.Co., Amsterdam, 1965.

[29] Rozenberg, G., and Salomaa, Arto, The Book of L, Springer Verlag, Heidelberg, 1986.

[30] Rozenberg, G., and Salomaa, Arto, The Mathematical Theory of L Systems Academic Press Inc., 1980

[31] Shanin, N.A., Constructive Real Numbers and Constructive Function Spaces AMS, Providence, Rhode Island, 1968.

[32] Shanin, N.A., On the Constructive Interpretation of Mathematical Judge-

Extended Toplogy, Information Sciences, Vol 26, 1982.

[45] Finch, P.D., On the Structure of Quantum Logic, Jr. of Symbolic Logic Vol. 34, No.2, 1969.

[46] Foo, N.Y., Homomorphisms in the Theory of Modelling, Int. Jr. General Systems, 1979.

[47] Fujiwara, T., A Variation of Lyndon-Keisler Homomorphism Theorem and its Applications to Interpolation Theorem, Jr. Math. Soc. Japan, Vol 25, No.1, March 1960.

[48] Gunson, J., On the Algebraic Structure of Quantum Mechanics, Commun. Math. Phys. 6, 1967.

[49] Hammer, P.C., General Topology, Symmetry and Convexity, Wis. Academy of Science, Arts and Letters, Vol 44, 1955.

[50] Hammer, P.C., Extended Toplogy: Set-Valued Set-Functions, Nieuw Archief voor Wiskunde (3), X, 1962.

[51] Henkin, L., An Extension of the Craig-Lyndon Interpolation Theorem, Jr. of Symbolic Logic, Vol 28, No.3, Sept. 1963.

[52] Jauch, J.M., and Piron, C., On the Structure of Quantal Proposition systems, Helvestia Physica Acta, Vol 42, 1969.

[53] Keisler, H.J., Theory of Models with Generalized Atomic Formulas, Jr. of Symbolic Logic, Vol 25, No.1, March 1960.

[54] Kleijn, H.C.M., Selective Substitution Grammar, Ph.D. Thesis, University of Leiden, The Netherlands.

[55] Kleijn, H.C.M., Some Basic Ideas of Selective Substitution Grammar, Lecture Notes in Computer Science, Vol. 281, Springer Verlag, 1987.

[56] Lindenmayer, A., Models for Multicellular Development: Characterization, Inference and Complexity of L-Systems, Lecture Notes in Computer Science, Vol. 281, Springer Verlag, 1987.

[57] Lyndon, R.C., An Interpolation Theorem in the Predicate Calculus, Pacific Jr. Math., 9, 1959.

[58] Lyndon, R.C., Properties Preserved under Homomorphisms, Pacific Jr. Math., 9, 1959.

[59] Markov, A. A., On the Language $Я_0$, Soviet Math. Dokl. Vol. 15 No.1, 1974.

[60] Markov, A. A., On the Language $Я_1$, Soviet Math. Dokl. Vol. 15 No.1, 1974.

[61] Markov, A. A., On the Language $Я_2$, Soviet Math. Dokl. Vol. 15 No.1, 1974.

[62] Markov, A. A., On the Language $Я_3$, Soviet Math. Dokl. Vol. 15 No.1, 1974.

[63] Markov, A. A., On the Language $Я_4$, $Я_5$,....... , Soviet Math. Dokl. Vol. 15 No.1, 1974.

[64] Markov, A. A., On the Language $Я_{\omega}$, Soviet Math. Dokl. Vol. 15 No.1, 1974.

[65] Markov, A. A., On the Language $Я_{\omega|}$, Soviet Math. Dokl. Vol. 15 No.1, 1974.

[66] Markov, A. A., On the Completeness of the Classical Predicate Calculus in Constructive Mathematical Logic, Soviet Math. Dokl. Vol. 15 No.1, 1974.

[67] Markov, A. A., Normal Algorithms which Compute Boolean Functions, Soviet Math. Dokl., 5, 1964.

[68] Shanin, N.A., Role of a Notion of Algorithm in the Arithmetic Language Semantics, Lecture Notes in Computer Science, Vol. 122, Springer Verlag, 1981.

[69] Schützenberger, M.P., On the Definition of a Family of Automata, Inf. and Control 4, 1961.

[70] Schütte, Kurt, Der Interpolationssatz der intuitionistischen Prädikatenlogik, Math. Annalen 148, 1962.

[71] Scott, D., Suppes, P., Foundational Aspects of the Theory of Measurement, Jr. Symbolic Logic, Vol. 23, No.2, January 1958.

[72] Sinha, V.P.,

(1). Certain Logical Considerations in Finite Discrete System Theory,

First International Workshop on Fault Detection and Spectral Techniques, Oct. 12-14, 1983, Boston university.

(2). On the Logic of Signal Processing, Second International Workshop on Spectral Techniques, Oct.5-7, 1986, École Polytechnique, Montreal.

[73] Steiglitz, K., The Equivalence of Digital and Analog Signal Processing, Inf. and Control 8, 1965.

[74] Thampuran, D.V., Extended Topology: Filters and Convergence - I, Math. Annalen, Vol. 158, 1965.

EE-1990-D-RAJ-STU